Copyright © Signalogic 1994-2013

DirectCore Software

DirectCore Software and API Reference Guide

Revision 3.03
June 2013

Signalogic, Inc.
9617 Wendell Rd.
Dallas, Texas 75243
Tel: 214-349-5551
Fax: 214-343-0069

The software described in this document is furnished under license agreement. The software may be used or copied only under the terms of this license agreement (see section 3.1.1).

Copyright © 1994-2013 Signalogic, Inc. All rights reserved.
No part of this manual may be photocopied or reproduced in any form without prior written consent from Signalogic, Inc.
U.S. Government: If Licensee is acquiring the software on behalf of any unit or agency of the United States Government, the following shall apply:

(a) for units of the Department of Defense:
RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data Clause at DFARS 252.227-7013.

(b) for any other unit or agency:
NOTICE -- Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding use, reproduction and disclosure are as set forth in Clause 52.227-19(c)(2) of the FAR.

Contractor/manufacturer is Signalogic, Inc., 9617 Wendell Rd., Dallas, TX 75243.

Signalogic, DirectDSP, and DSPower are registered trademarks, and DirectCore, CIM, and Real-Time Convolver are trademarks of Signalogic, Inc.
MATLAB is a registered trademark of The MathWorks, Inc.
National Instruments is a registered trademark, and NIDSP is a trademark of National Instruments, Inc.
Microsoft is a registered trademark, and MS-DOS, Windows, and Visual C are trademarks of Microsoft Corporation.
TMS320xx, TMS320C5000, TMS320C6000, SYSBIOS, and Code Composer Studio are trademarks of Texas Instruments.
Hyperception is a registered trademark of Hyperception, Inc. (acquired by National Instruments in 2004).
Analog Devices, AT&T, Freescale, and NEC are registered trademarks of their respective holders. ADSP-21xx, ADSP-2106x, WE DSP32x, DSP56xxx, DSP9600x, 77220, are trademarks of their respective holders.
PC is a trademark of IBM Corporation.
All other trademarks belong to their respective holders.

Signalogic, Inc.
9617 Wendell Rd.
Dallas, TX 75243
214-349-5551

Signalogic DirectCore Software Reference Guide, Created November 2003
Jeff Brower, Chris Johnson, Vikram Ragukumar, Yahia Tachwali, Yolanda Lei, Ru Chen, Varsha Shamabhat, Hanh Nguyen.

Copyright © Signalogic 1994-2013

Contents

1 Overview 1

1.1 Supported Hardware 1
1.2 Product History 2
1.3 Product Support 2
1.4 Architecture Overview 3

2 API Reference 6

Overview and Conventions 6
2.1 API Quick References 8
2.2 Library Functions 14
2.2.1 Hardware Library Functions 15
2.2.2 File Library Functions 81
2.2.3 Engine Manager Functions 81
2.2.4 Hardware Manager Functions 101
2.3 About Core Lists 121
2.3.1 Abstraction of Cores 122
2.3.2 API Core List Parameters 122
2.4 RTAF Properties 124
2.5 DSP Engine Functions 147

3 Waveform File Formats 149

Overview 149
3.1 Hypersignal Waveform File Format 150
3.2 Microsoft .WAV Waveform File Format 152
3.3 Acoustic/Audio Waveform File Formats 154
3.3.1 Techron .TDS File Format 155
3.3.2 Ariel SYSid File Format 156
3.3.3 Micro Technology Unlimited .SF File Format 157

4 Definitions 158

Overview 158
4.1 Structure Definitions 158
4.1.1 HWLISTENTRY Structure 159
4.1.2 CONVERSIONINFO Structure 162
4.1.3 DSPDATARDYMSGINFO Structure 165
4.1.4 ERRORMSGINFO Structure 167
4.2 Constant Definitions 168

5 Messages 169

Overview 169
5.1 Library Messages 170
5.2 DSP Engine Messages 187

6 Error and Status Conditions 197

Overview 197
6.2 DLL Error and Status Conditions 200
6.2.1 General DLL Error and Status Conditions 201
6.2.2 Engine Manager Error and Status Conditions 203
6.2.3 Hardware Manager Error and Status Conditions 205
6.2.4 Hardware Library Error and Status Conditions 207
6.2.5 Signal Processing Library Error and Status Conditions 210
6.3 DSP Engine Error and Status Conditions 211
6.3.1 Hypersignal Error and Status Conditions 212
6.3.2 MATLAB Error and Status Conditions 213

Glossary of Terms 214

 Copyright © Signalogic 1994-2013

1 Overview

DirectCore™ software provides API access to multicore CPU and DSP hardware, with emphasis on the following areas:

Ease-of-Use

- Standardized and well-organized API, developed and refined over many years of support for numerous types of hardware
- Detailed and extensive documentation

Performance

- High bus transfer rates
- Synchronous and asynchronous transfers. Driver supports “fire and forget” transfers with deferred notification
- Card-to-card bus mastering transfers
- Optimized transfer parameters, including buffer sizes, contiguous memory, word widths, and CPU and/or DSP external memory and cache considerations

Multicore Functionality

- Abstraction of number of cards, number of cores
- Independent core access, group/list-of-cores access

Accelerator Support

- Functionality required when hardware is used as an OpenMP accelerator from Linux C/C++ applications
- Thread notification, resource management framework,

TI Compatibility

- Executable and library code download, including support for COFF and ELF formats
- Symbol look-up; determine by name physical address of C code symbols (variables, arrays, structures, etc)

Debug

- A combination of probe points (device code events) and memory transfers can be used to avoid tedious JTAG level debug when possible
- A number of test and measurement utilities provided, including driver-only, library + driver, test and demo reference source codes

1.1 Supported Hardware

DirectCore includes a Linux LKM (loadable kernel mode) driver for supported multicore CPU and DSP hardware, including various PCI and PCIe cards, ATCA boards, and PC104 boards. The table in Figure 1-1 below gives a partial list of supported hardware.

Card	Manufacturer	Device Type	Form-Factor / Format	DirectCore Functionality	OpenMP Accelerator Support
DSPC-8681E (SigC6678-32)	Advantech	TI C6678	PCIe ½ length, full height card	Full	Y

DSPC-8682E (SigC6678-64) [1]	Advantech	TI C6678	PCIe full-length, full height card	Full	Y
DSPA-8901 [2]	Advantech	TI C6678	ATCA [2]	Full	Y
SigC641x	Signalogic	TI C6415	PTMC module PCI or PCIe ½ length, full-height card	Full, but some limitations on multicore APIs	Y
SigC5561	Signalogic	TI TNETV3010	PTMC module PCI or PCIe ½ length, full-height card	Partial	N
SigC5510	Signalogic	TI C5510	PC104, SIMM and DIMM module	Most, but some limitations on multicore APIs	N
SigC54xx	Signalogic	TI C5416	PC104	Most, but some limitations on multicore APIs	N
Legacy Hardware [3]	Various	TI C54x TI C4x TI C3x Freescale 563xx Analog Devices 21xx Analog Devices 2106x Lucent DSP32C	ISA PC104 PCI	Limited functionality	N

Figure 1-1 Partial list of hardware supported by DirectCore software

Notes

[1] Under test, but not released yet.

[2] Planned. Inquire for availability.

[3] For a list of legacy hardware supported by DirectCore software, see http://signalogic.com/hwlist.htm

1.2 Product History

The DirectCore™ software product was first introduced in 1994 as DirectDSP® software, with support for many types of DSP hardware and different DSP vendors added and deprecated over the years. In 2009 the product was renamed to DirectCore to reflect the increasing importance of the product's underlying support for advanced software products including SigIMS and CIM™ (Compute Intensive Multicore) software, with the latter providing OpenMP accelerator functionality for multicore CPU devices such as the Texas Instruments C66x series.

1.3 Product Support

The DirectCore software product includes 60-day of engineering staff technical support, typically via e-mail and phone. Additional technical support and maintenance agreements are available.

Copyright © Signalogic 1994-2013

1.4 Architecture Overview

Below is software architecture diagram showing DirectCore software's central role in a system including application software, an optional API layer for compute-intensive applications, optional CIM software, and multicore CPU or DSP software.

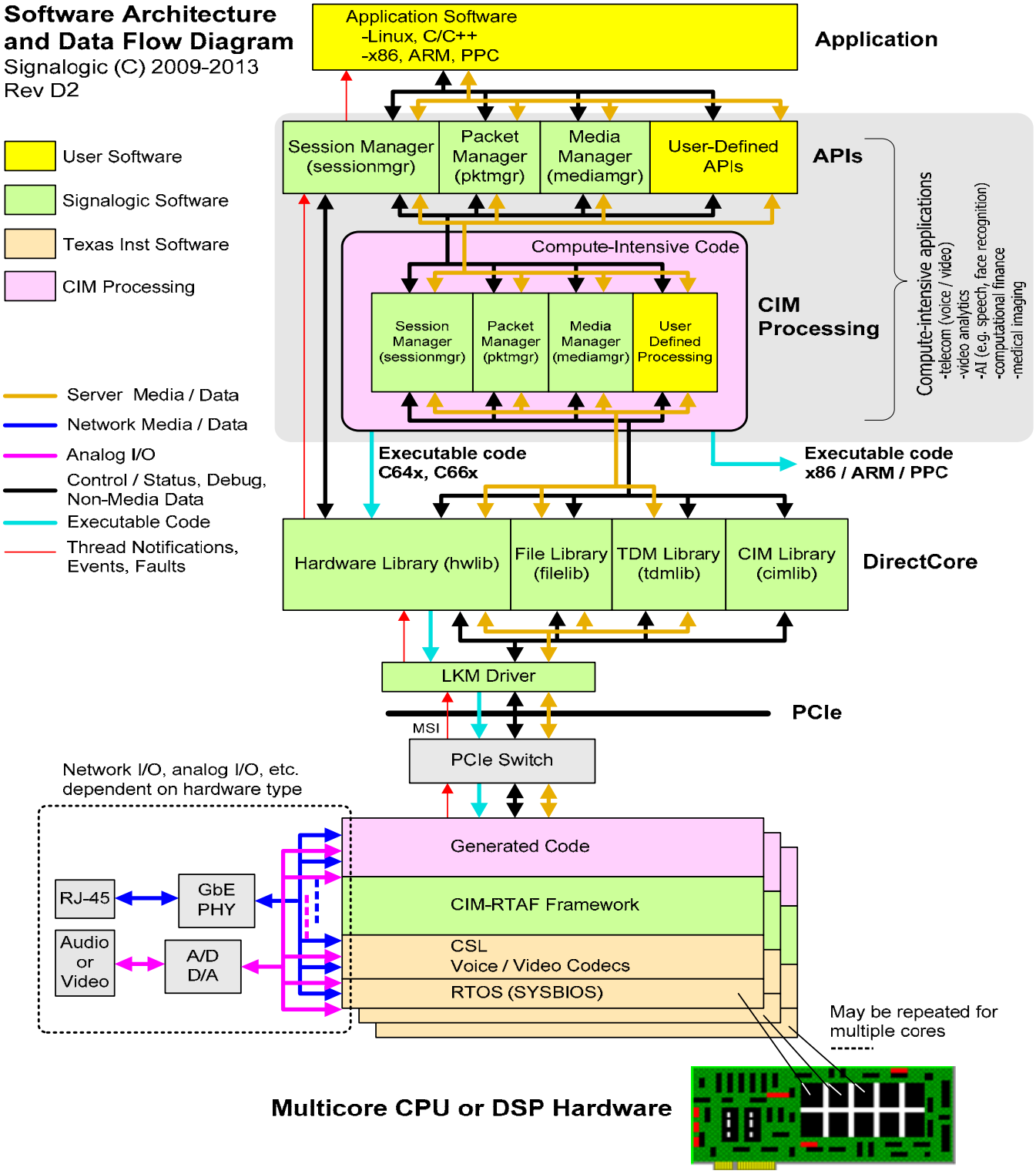

Figure 1-2 System Architecture Diagram, with DirectCore providing underlying hardware API

Figure 1-2 below shows a “minimal architecture”, with application API and CIM software removed, and user application programs directly accessing DirectCore APIs.

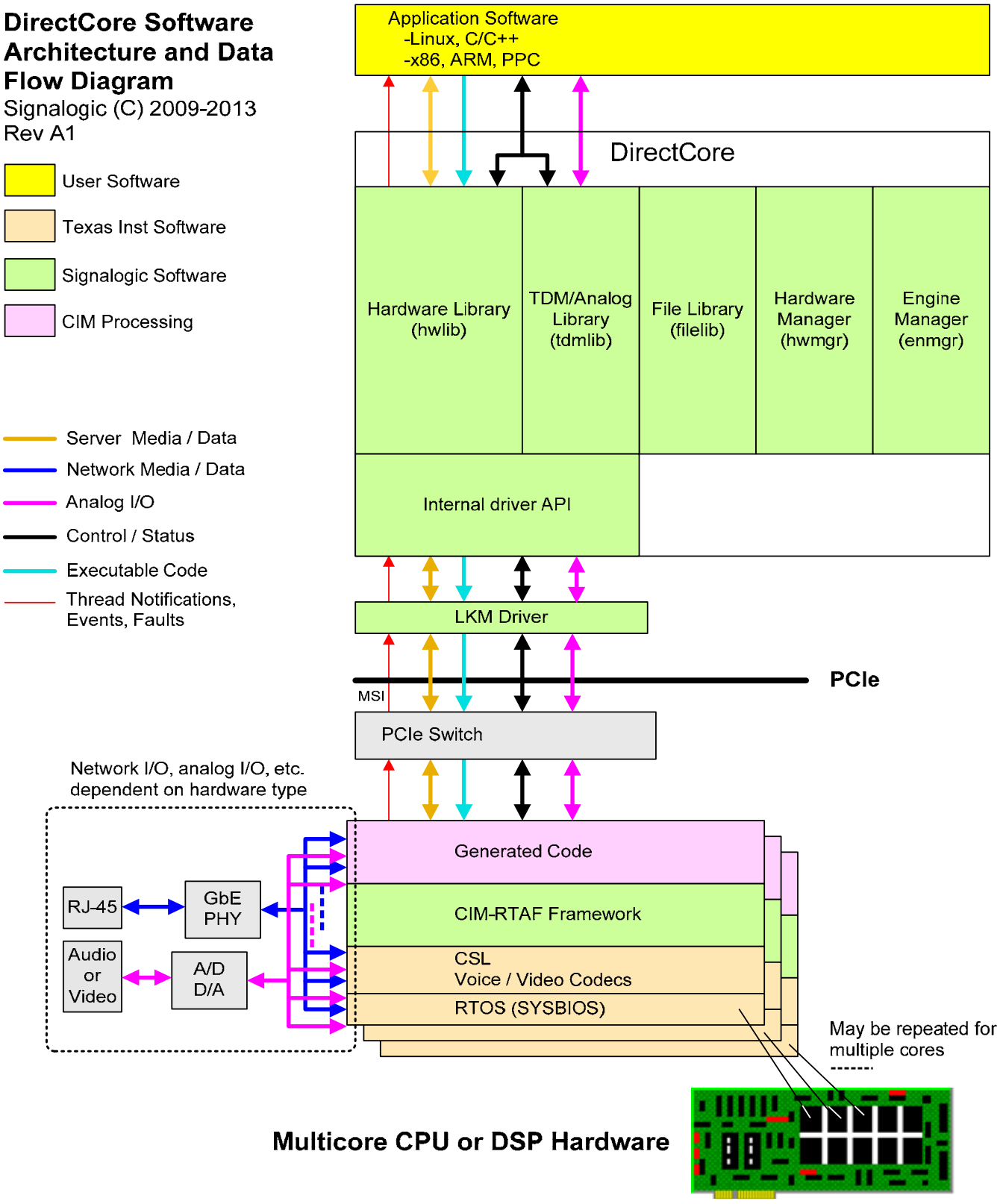

Figure 1-3 “Minimal Architecture” Diagram, with application layer calling DirectCore APIs directly

 Copyright © Signalogic 1994-2013

Architecture Diagram Notes

Following are some notes about the above software architecture diagrams.

1. In the minimal architecture case, note that without CIM software, multicore CPU or DSP executable code must be generated externally and independently, then downloaded "manually" to multicore CPU or DSP hardware from the host application, using DirectCore APIs.
2. "Executable Code x86, ARM, PPC" refers to host executable code; i.e. code running on CPU cores native to the server or other host application platform.
3. Hardware Manager and Engine Manager modules are not shown Figure 1-2, but they are part of DirectCore. They are shown in Figure 1-3.

DirectCore Components

As shown in Figure 1-3 above, DirectCore software is composed of several modules, or libraries, as described in Table 1-1 below.

Library	Filename(s)	Comments
Hardware API	hwlib.so hwlib.dll	Access, data transfer, and control & status for device and device memory Card resource allocation and management
File / Waveform API	filelib.so filelib.dll	Management of waveform files and conversion between file formats
TDM / Analog API	tdmlib.so	Access, data transfer, and control & status for TDM and analog resources on the card, if any ("analog" includes audio and video)
CIM API	cimlib.so	Library used by CIM processing, including resource management. Only applicable if CIM software is used.
Hardware Manager	hwmgr.so hwmgr.dll	Visual hardware management, mostly not used in Linux software
Engine Manager	enmgr.so enmgr.dll	Engine management, mostly not used in Linux software

 Copyrght © Signalogic 1994-2013

2 API Reference

Overview and Conventions

The API Reference section provides technical descriptions of functions available in DirectCore software.

API Parameters

General parameters used in DirectCore API calls are shown in the table below.

Parameter	Description	C Language Equivalent
BOOL	boolean	unsigned char
BYTE	byte	unsigned char
DWORD	unsigned 32-bit word (double word)	unsigned int
QWORD	unsigned 64-bit word (quad word)	unsigned long long
HCARD	handle to multicore CPU or DSP hardware (e.g. PCIe card)	unsigned int
int	signed 32-bit integer	int long
LPCSTR	pointer to zero-terminated string	const char*
PMEM	pointer to user space memory used in multicore CPU or DSP hardware data transfers	void* void far*
UINT	unsigned 32-bit integer	unsigned int
WORD	unsigned 16-bit word (single word)	unsigned short int

Formatted String Parameters

There are a few API functions where string input parameter (LPCSTR in the above table) expect a certain format in order to specify multiple sub parameters in the string. The convention used in the API documentation for text parameters within a formatted string is follows:

- Required entry is formatted as bold
- Optional parameters are enclosed in square brackets
- A special case arises when analog channel inputs are being referred to. The convention used is "**an**", where n represents a number. For instance, a0 would indicate channel 0, a1 would indicate channel 1, etc.

For example a string parameter entry given as:

*[path]**filename**[ext]*

 Copyright © Signalogic 1994-2013

would mean the filename entry is required, but the path and extension are optional.

Formatted String Numerical Parameters

Some formatted strings also allow numerical parameter entry. If the entry required is a number, then this field contains "n" followed by the required units for the parameter. This is followed by the possible range. Any special details are mentioned later enclosed in-between /* and */. A typical entry would look like this:

n <units>; <range allowed>; /* special cases */

For example:

n *sec; >0*; /* 0 = All */

See Also

API Quick Reference

Function Categories

Library Functions

DSP Engine Functions

2.1 API Quick References

The tables below provide quick-references for DirectCore library functions, including a functional grouping for Hardware Library APIs, and alphabetical lists for (i) Hardware Library APIs, and (ii) overall DirectCore APIs.

The quick reference tables include page number links to the corresponding API descriptions, which contain parameter information and calling instructions, comments, related functions, and examples.

Hardware Library API Quick Reference, by Functional Group

Group	Function Name	Attributes	Page
Card Management	DSAssignCard		17
	DSFreeCard		26
	DSGetCardClass		32
	DSGetCardInfo		32
	DSGetMemArch		40
	DSGetMemSize		40
	DSGetWordLength		42
	DSGetCardBaseAddr	D	33
	DSGetCardBusType	D	30
	DSSetCardBaseAddr	D	66
	DSSetCardBusType	D	66
Card Control and Status			
	DSInitCard		45
	DSGetCoreList		36
	DSResetCores, DSResetCard		61, 53
	DSResetDevices	V	63
	DSRunCores, DSRunCard		65, 62
	DSRunDevices	V	67
	DSSetCoreList		70
	DSDisableCard		25
	DSHoldCores, DSHoldCard	D	44, 41
Data and Program Memory and Register Access			
	DSGetProperty		34
	DSSetProperty		50
	DSReadMem		37
	DSReadMemEx		58
	DSWriteMem		51

	DSWriteMemEx		79
	DSReadCardReg	D	53
	DSWriteCardReg	D	74
Program Download and Symbol Lookup			
	DSLoadFileCard		46
	DSLoadFileCores		48
	DSGetSymbolAddr		39
Notification and Callback			
	DSRegisterCallbackFunc		
	DSUnregisterCallbackFunc		
	DSWaitForBuffer		69
	DSWaitForFlag		71
Waveform and Analog	DSAcquireWvfrmFile		17
	DSCalcSampFreq		19
	DSGenerateWvfrmFile		29
	DSDSPToIEEE		28
	DSIEEEToDSP		47

Attributes

D = Deprecated (typically applicable only to legacy hardware).

V = Significantly variable behavior, depending on hardware type.

Hardware Library API Quick Reference, Alphabetic

Function Name	Page
DSAcquireWvfrmFile	17
DSAssignCard	17
DSCalcSampFreq	19
DSCancelWaitBuffer	23
DSCancelWaitFlag	23

DSDisableCard	25
DSDSPToIEEE	28
DSFreeCard	26
DSGenerateWvfrmFile	29
DSGetCardBaseAddr	33
DSGetCardBusType	30
DSGetCardClass	32
DSGetCardInfo	32
DSGetCoreList	36
DSGetProperty	34
DSGetMemArch	40
DSGetMemSize	40
DSGetSymbolAddr	39
DSGetWordLength	42
DSHoldCard	41
DSHoldCores	44
DSInitCard	45
DSIEEEToDSP	47
DSLoadFileCard	46
DSLoadFileCores	48
DSReadCardReg	53
DSReadMem	37
DSReadMemEx	58
DSRegisterCallbackFunc	
DSResetCard	53
DSResetCores	61
DSResetDevices	63
DSRunCard	62
DSRunCores	65
DSRunDevices	67
DSSetCardBaseAddr	66
DSSetCardBusType	66
DSSetCoreList	70
DSSetProperty	50
DSUnregisterCallbackFunc	
DSWaitForBuffer	69
DSWaitForFlag	71

DSWriteCardReg	74
DSWriteMem	51
DSWriteMemEx	79

Overall DirectCore API Quick Reference, Alphabetic

Function Name	Page
DSAcquireWvfrmFile	17
DSAddEngMgrStatusLine	82
DSAssignCard	17
DSCalcSampFreq	19
DSCancelWaitBuffer	23
DSCancelWaitFlag	23
DSDisableCard	25
DSDSPToIEEE	28
DSEngineOpen	83
DSEngineClose	84
DSExecEngineFunction	85
DSFreeCard	26
DSGenerateWvfrmFile	29
DSGetCardBaseAddr	33
DSGetCardBusType	30
DSGetCardClass	32
DSGetCardInfo	32
DSGetCoreList	36
DSGetEngineErrorStatus	88
DSGetEnginePath	89
DSGetEngineUsage	90
DSGetEngineState	90
DSGetEngMgrErrorStatus	91
DSGetProperty	34
DSReadMem	37
DSReadMemEx	58
DSRegisterCallbackFunc	
DSGetMemArch	40
DSGetMemSize	40
DSGetHWMgrCurVal	101

DSGetHWMgrEntry	102
DSGetHWMgrEntryIndex	104
DSGetHWMgrErrorStatus	105
DSGetHWMgrNumEntries	106
DSGetHWMgrVar	107
DSGetHWSetupFile	109
DSGetSymbolAddr	39
DSGetWordLength	42
DSHideEngMgrStatusWindow	93
DSHoldCard	41
DSHoldCores	44
DSInitCard	45
DSIEEEToDSP	47
DSLoadFileCard	46
DSLoadFileCores	48
DSReadCardReg	53
DSReadHWSetupFile	109
DSRegisterCallbackFunc	
DSRegisterEngineMsgWnd	94
DSResetCard	53
DSResetCores	61
DSResetDevices	63
DSRunCard	62
DSRunCores	65
DSRunDevices	67
DSSendEngineCommand	95
DSSetCardBaseAddr	66
DSSetCardBusType	66
DSSetCoreList	70
DSSetEngineTemplatePath	97
DSSetEngineWaveformPath	98
DSSetHWMgrEntry	113
DSSetHWMgrCurVal	111
DSSetHWMgrVar	114
DSSetHWSetupFile	116
DSSetProperty	50
DSShowEngMgrStatusWindow	98

DSShowHardwareSelectorDlg	116
DSUnregisterCallbackFunc	
DSUnregisterEngineMsgWnd	99
DSWaitForBuffer	69
DSWaitForFlag	71
DSWriteCardReg	74
DSWriteHWSetupFile	116
DSWriteMem	51
DSWriteMemEx	79

 Copyrght © Signalogic 1994-2013

2.2 Library Functions

Library functions have been further subdivided into the following categories:

- Hardware Library Functions (hwlib)
- File Library (filelib)
- CIM Library (cimlib – ***only used if higher level CIM software is included in the system. See the Architecture diagram in Figure 1-2 above)***)
- TDM Library (tdmlib)
- Engine Manager Functions (enmgr)
- Hardware Manager Functions (hwmgr)

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

For a complete listing of all DirectCore software functions, see the API Reference, above.

NOTE: In the function descriptions listed below, knowledge of type definitions declared in standard Linux and Microsoft include files is assumed. For example, LPCSTR is defined as const char*, UINT is defined as unsigned int, etc.

Related Users Guide Chapters

Using Engine Manager

Using Hardware Manager

Using DirectCore Software

2.2.1 Hardware Library Functions

Below are listed functions accessible in the Hardware Library API. For information on how to include Hardware Library APIs in Linux applications, see example programs and make files under the ../DirectCore/apps subfolder, as shown in figure 2-1 below.

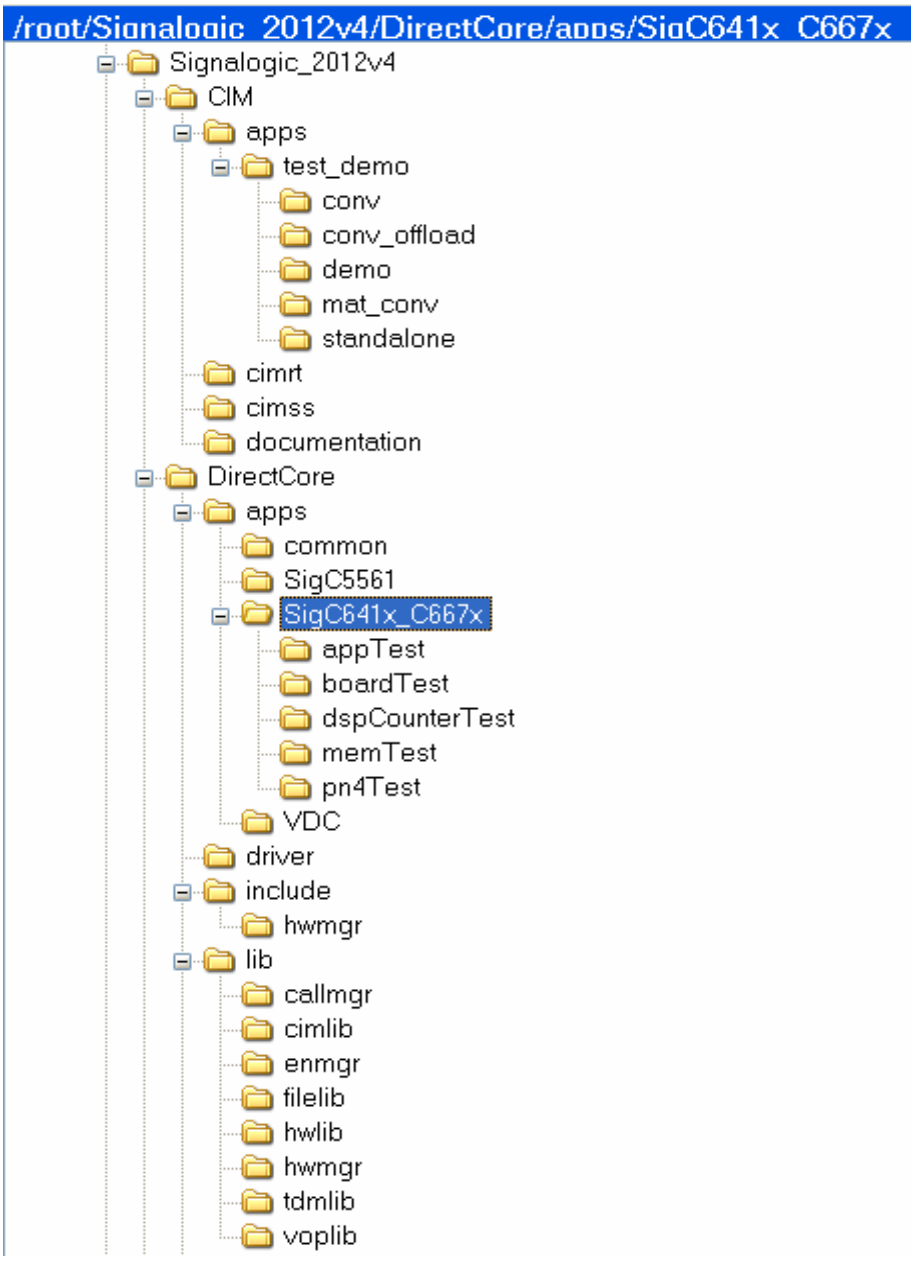

Figure 2-1 DirectCore software Linux install subfolder hierarchy

For information on how to include Hardware Library in Windows applications (hwlib.dll or hwlib32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic Software Users Guide section 4.2, Creating Programs for DirectCore.

Copyrght © Signalogic 1994-2013

Hardware Library Functions

DSAcquireWvfrmFile
DSAssignCard
DSCalcSampFreq
DSCancelWaitBuffer
DSCancelWaitFlag
DSDisableCard
DSDSPToIEEE
DSFreeCard
DSGenerateWvfrmFile
DSGetCardBaseAddr
DSGetCardBusType
DSGetCardClass
DSGetCardInfo
DSGetProperty
DSGetMemArch
DSGetMemSize
DSGetWordLength
DSGetSymbolAddr
DSHoldCard
DSHoldCores
DSInitCard
DSIEEEToDSP
DSLoadFileCard
DSLoadFileCores
DSReadCardReg
DSReadMem
DSReadMemEx
DSResetCard
DSResetCores
DSResetDevices
DSRunCard
DSRunCores
DSRunDevices
DSSetCardBaseAddr
DSSetCardBusType
DSSetProperty
DSWaitForBuffer

DSWaitForFlag

DSWriteCardReg

DSWriteMem

DSWriteMemEx

DSAcquireWvfrmFile

UINT DSAcquireWvfrmFile (hCard, pConversionInfo)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
PCONVERSIONINFO pConversionInfo	// pointer to CONVERSIONINFO struct containing data acquisition or stimulus & response information

The DSAcquireWvfrmFile function performs a data acquisition or stimulus & response operation, storing data from one or more analog input channels on the specified multicore CPU or DSP/analog hardware continuously to waveform file.

Parameter	Description
hCard	Handle to multicore CPU or DSP/analog hardware previously obtained from DSAssignCard call
pConversionInfo	Pointer to CONVERSIONINFO structure containing information about the data acquisition or stimulus & response operation to be performed

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the operation failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Because this is an asynchronous function, calling applications are notified of any problems encountered (waveform file errors, compile errors, etc.) via a callback function. For Linux applications, notification is sent via a signal to a function previously registered with the DSRegisterCallbackFunc API. For Windows applications, notification is sent via a WM_DSPENGINE_PROGSTATUS message, and application code must have a window message handler set up to receive such PROGSTATUS messages.

C/C++ Example

```
HLOCAL hConvInfo;
PCONVERSIONINFO  pConversionInfo;

// create CONVERSIONINFO object memory; use zero-initialization to set most parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE | LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

// file into which to acquire data (including path information)

lstrcpy(pConvInfo->szFilename, szWaveformFilename);

// setup for three (3) channels, init channel list (example is sequential, but could be
// arbitrary channel order)
```

```
pConvInfo->NumChan = 3;          // three channels
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz is desired sampling rate
pConvInfo->NumSamples = 100000L;  // acquire 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO

// start acquisition

DSAcquireWvfrmFile(hCard, pConvInfo);

// free and delete hConvInfo memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

In the above example, numerous elements of the CONVERSIONINFO structure have been left at their default (or zero) values. For instance, no trigger information has been set, digital offset and scale factors are not active, and the default break mode (store acquired data if user-interrupted early) is in effect.

Note also that the stored data file will actually represent 300,000 total samples, because three channels were specified.

See Also

DSGenerateWvfrmFile

WM_DSPENGINE_PROGSTATUS

DSAssignCard

DSExecEngineFunction

Related Users Guide Topics

section 4.5, Acquiring Data Using DirectCore Software

Program Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.1.1, Engine and Card Handles in DirectCore Software

section 7.3, Executing Functions in DSP Engines

 Copyright © Signalogic 1994-2013

DSAssignCard

HCARD DSAssignCard (hEngine,
pszCardDesignator,
uBusType,
wIOBaseAddr,
wMemBaseAddr,
fEnableTalker,
uNumDevices)

HENGINE hEngine	// engine handle. Should be set to NULL for Linux applications (not currently used in Linux applications)
LPCSTR pszCardDesignator	// pointer to card designator string
UINT uBusType	// bus type
WORD wIOBaseAddr	// I/O base address
WORD wMemBaseAddr	// memory base address (if any)
BOOL fEnableTalker	// flag to enable/disable "narrow pipe" type of data transfer between host code and the multicore CPU or DSP hardware (for which code resident on the hardware is sometimes referred to as a "talker" program)
UINT uNumDevices	// number of active devices on the multicore CPU or DSP hardware

The DSAssignCard function allocates and assigns a handle for the specified multicore CPU or DSP/analog hardware type. Some important hardware configuration parameters can be initialized at the same time. The hardware type is specified using a card designator string.

The returned handle can be used in numerous other Hardware Library function calls.

Parameter	Description
hEngine	Should be set to NULL for Linux applications. For Windows applications, should be an engine handle previously obtained from a DSEngineOpen call
pszCardDesignator	Pointer to null-terminated card designator string
uBusType	Should be set to NULL (use default capability of the hardware) for PCI and PCIe card type specified by pszCardDesignator. For older / legacy hardware, uBusType specifies the bus type of the multicore CPU or DSP/analog indicated by pszCardDesignator. Possible bus types are given by the values below; a NULL value indicates the default Bus Type in the Hardware Setup File should be used

Value	Meaning
DS_AC_PCXT	PC-XT bus type (deprecated)
DS_AC_PCAT	PC-AT bus type (deprecated)
DS_AC_ENABLE80X86INST	Enable 80x86 instruction set operations (deprecated)
DS_AC_PCIEX1	PCIe bus type, 1 lane
DS_AC_PCIEX4	PCIe bus type, 4 lanes
DS_AC_PCIEX8	PCIe bus type, 8 lanes
DS_AC_PCIEX16	PCIe bus type, 16 lanes

wIOBaseAddr For PCI and PCIe hardware, wIOBaseAddr should be set to NULL (not used) or set with attributes shown below. For older / legacy hardware, wIOBaseAddr specifies the I/O base address of the multicore CPU or DSP/analog hardware indicated by pszCardDesignator. A NULL value indicates the default I/O Base Address in the Hardware Setup File should be used

Value	Meaning
Max bus transfer buffer size	A value from 128 to 32768 that specifies the maximum size of PCI / PCIe transfer buffers (in bytes). For the default value of zero (NULL), the max bus transfer buffer size is 1024 bytes

wMemBaseAddr For PCI and PCIe hardware, wMemBaseAddr should be set to NULL (not used) or set with attributes shown below. For older / legacy hardware, wMemBaseAddr specifies the memory base address the multicore CPU or DSP/analog hardware indicated by pszCardDesignator. A NULL value indicates the default Memory Base Address in the Hardware Setup File should be used

Value	Meaning
DS_AC_RESET_CPUS_SIMULTANEOUS	Causes multicore CPU or DSP devices to be reset simultaneously during DSResetCard, DSResetCores, or DSResetDevices API. See comments below
DS_AC_MEMMODE_DIRECT	Causes “direct memory access” mode to be used in the driver for multicore CPU or DSP hardware driver
DS_AC_MEMMODE_MASTER_NOWAIT	Causes asynchronous bus mastering mode to be used in the driver for multicore CPU or DSP hardware
DS_AC_MEMMODE_MASTER_WAIT	Causes synchronous bus mastering mode to be used in the driver for multicore CPU or DSP hardware
DS_AC_MEMMODE_SLAVE_WAIT	Deprecated, should not be used
DS_AC_MEMMODE_MASK	Can be used to separate (or mask) DS_AC_MEMMODE_xxx values from other values when manipulating this attribute
DS_AC_CORELIST64	Default setting for treatment of API core list parameters. Indicates that core list parameters are a single QWORD or less (support for maximum of 64 cores). Leaving this attribute as NULL gives the same result
DS_AC_CORELISTEXTENDED	Indicates that API core list parameters are pointers to a list of up to 256 QWORDs (maximum of 16384 cores)

fEnableTalker Flag to enable/disable “narrow pipe” type of data transfer between

	host code and the multicore CPU or DSP hardware (for which code resident on the hardware is sometimes referred to as a "talker" program). Should be set to FALSE in most cases. Should only be set to TRUE for some types of legacy hardware. See the Glossary definition of Talker for more information
uNumDevices	Sets the active number of devices (multicore CPUs, DSPs, etc) on the hardware specified by pszCardDesignator. May be set to NULL to use the default number of devices on the hardware. Can be set to some number less than the default number of devices on the hardware for certain situations, such as debug / test, reduced power consumption, etc.

Returns

The hCard return value is a non-zero multicore CPU or DSP/analog hardware handle if successful. Otherwise, the return value is NULL, indicating the DSAssignCard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

The hCard return value should be used in subsequent DirectCore APIs.

Comments

The pszCardDesignator parameter is a string value which specifies the multicore CPU or DSP/analog hardware type. The Hardware Setup File maintained by Hardware Manager contains the current list of valid card designators. For Windows or GUI based applications, this value may also be conveniently chosen by an operator using the Hardware Selector dialog box. For more information about card designators, see Card Designator Field in Hardware Setup File, in Signalogic Software Users Guide section 8.3.1, Hardware Setup File Fields.

DSAssignCard sbould be called once for each instance of multicore CPU or DSP hardware in the system. The DS_GCI_NUMDEVICES and DS_GCI_NUMCORESPERDEVICE attributes can be used with the DSGetCardInfo API to determine hardware resources available after calling DSAssignCard.

For more information about the DS_AC_CORELISTxxx attribute values, see section 2.3.2 below, API Core List Parameters.

A value of NULL can be given for the uBusType, wIOBaseAddr, and wMemBaseAddr parameters to cause the current Bus Type, I/O Base Address, or Memory Base Address values in the Hardware Setup File to be substituted for these parameters. These values can also be changed at any time using the DSSetCardBusType and DSSetCardBaseAddr functions. PCI and PCIe types of multicore CPU or DSP/analog hardware do not require a memory base address. See Related Users Guide Topics below for more information. For legacy hardware types, see also Signalogic Software System Guide section 4.3.2, Memory-Mapped Transfer Gotchas.

C/C++ Example

```
HENGINE hEngine = NULL;  // set to NULL for Linux applications
UINT uBusType1 = DS_AC_PCIEX8;
UINT uBusType 2 = DS_AC_PTMC;
LPCSTR szCardDesignator1 = "SigC6678-32";
LPCSTR szCardDesignator2 = "SigC641x";

hCard = DSAssignCard(hEngine, szCardDesignator1, uBusType1, NULL, NULL, FALSE, NULL);

// if attempt fails, try a different card in the system
if (hCard == NULL) hCard = DSAssignCard(hEngine, szCardDesignator2, uBusType2, NULL, NULL, FALSE, NULL);

if (hCard == NULL) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
```

```
}

if (hCard != NULL && !DSInitCard(hCard)) {

    // display message to user that card is not installed correctly or not
    // responding
}
```

Notes About Above Source Code Example

1) Normally a second attempt would not be made for a different type of hardware; i.e. host application code would be aware of what hardware is in the system and any failure of DSAssignCard would be treated as an error. This is just for example purposes only.

2) In the source code example above, uBusType is set only as an example where, for some reason, the multicore CPU or DSP hardware type requires an explicit bus type setting. In the case of almost all hardware, uBusType can be set to NULL, and DSAssignCard will function correctly.

See Also

DSShowHardwareSelectorDlg

Hardware Selector

Hardware Setup File

DSSetCardBusType

DSSetCardBaseAddr

DSEngineOpen

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

section 8.2, Hardware Selector

section 8.3, Hardware Setup File

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

 Copyright © Signalogic 1994-2013

DSCalcSampFreq

DWORD DSCalcSampFreq (hCard,
SampFreqRequested,
wNumChan,
pwChanList,
pSampFreqActual)

```
HCARD  hCard              // multicore CPU or DSP/analog hardware handle
float  SampFreqRequested  // desired sampling frequency
WORD  wNumChan            // number of channels
WORD*  pwChanList         // pointer to channel list
float*  pSampFreqActual   // pointer to actual sampling frequency
```

Given a desired sampling frequency, the DSCalcSampFreq function calculates the actual sampling frequency, and corresponding mode or control word value, for the specified multicore CPU or DSP/analog hardware. In cases where the hardware does not support, or only partially supports, simultaneous sampling, the number of channels and channel list can also be specified.

The DWORD return value represents a mode or control word which can be sent to code executing on the multicore CPU or DSP/analog hardware, for use in programming the oncard analog I/O clock generator.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
SampFreqRequested	Desired sampling frequency (IEEE 32-bit floating-point value)
wNumChan	Number of channels; used only for hardware with analog I/O sampling which is not fully simultaneous
pwChanList	Pointer to channel list; must have as many elements as specified by wNumChan; used only for hardware with analog I/O sampling which is not fully simultaneous
pSampFreqActual	Pointer to IEEE 32-bit floating-point value which will contain actual sampling rate upon return

Returns

The return value is a non-zero mode or control word suitable for use by code executing on the multicore CPU or DSP/analog hardware to program oncard analog I/O sampling frequency clock generators. Otherwise, the return value is NULL, indicating the DSAssignCard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Most types of multicore CPU or DSP/analog hardware are capable of sampling input and output data at a limited range and/or number of discrete rates. The range and number vary tremendously from one type of hardware to another. One common example is hardware with sigma-delta converters offering 16 rates varying from 8 kHz to 48 kHz. Another common example is hardware with successive approximation converters offering 65535 different rates varying from 150 Hz to 200 kHz.

The wNumChan and pwChanList parameters are needed only for multicore CPU or DSP/analog hardware with analog I/O sampling which is not fully simultaneous. Otherwise, these parameters should be set to NULL.

Examples where wNumChan and pwChanList are required include hardware with one or more multiplexed analog inputs or outputs and hardware with "channel scan" or "channel group" analog inputs or outputs. In these cases, the DSP Engine uses the wNumChan and pwChanList parameters to calculate a mode or control word reflecting an "aggregate sampling rate", which is some multiple of the requested sampling rate. Note that the pSampFreqActual return value does not reflect this aggregate rate, but always indicates the actual sampling frequency that will be in effect for each channel. pwChanList is not required if the hardware has *only one* multiplexed converter, channel group, etc., in which case the aggregate rate is simply the actual rate multiplied by the number of channels.

The DWORD return value is a mode or control register word which can typically be used by code executing on the multicore CPU or DSP/analog hardware, or by the DSP Engine to access a host server clock-generator register. In almost all cases, there is usually one register on the multicore CPU or DSP/analog hardware which is responsible for sampling frequency clock generation. This may be an onchip timer, an external timer device, a PAL or PEEL, a CODEC control register, etc. Having this value calculated by the DSP Engine provides the main advantage of informing the operator or user of differences between desired and actual sampling rates *a priori*.

When using RTAF software functions or operating modes, the return mode or control word value can be written to the DSPROP_FSMODE location, and the value pointed to by pSampFreqActual can be written to the DSPROP_FSVALUE location.

A successful DSInitCard function must have been completed before DSCalcSampFreq can be called.

C/C++ Example

```
DWORD dwFsMode;
float ActualFs, DesiredFs;

DesiredFs  = 75000;  // desired sampling rate is 75 kHz
dwFsMode = DSCalcSampFreq(hCard, DesiredFs, NULL, NULL, &ActualFs);

DSSetProperty(hCard, DSPROP_FSMODE, dwFsMode);
DSSetProperty(hCard, DSPROP_FSVALUE, ActualFs);
```

In the above example, the wNumChan and pwChanList parameters have been ignored; these are only required for some hardware types with non-simultaneous sampling (e.g. inputs which are “scanned”, or sampled on a rotating basis, which repeats).

See Also

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSSetProperty

DSPROP_FSMODE

DSPROP_FSVALUE

DSGetCardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

 Copyright © Signalogic 1994-2013

DSCancelWaitBuffer

UINT DSCancelWaitBuffer (hCard,
wBufferNum)

```
HCARD  hCard        // multicore CPU or DSP/analog hardware handle
WORD  wBufferNum    // buffer number
```

The DSCancelWaitBuffer function cancels a pending DSP Engine wait-for-buffer request with the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
wBufferNum	Number of buffer to cancel wait request for

Returns

The return value is TRUE if the wait-for-buffer request was canceled successfully. Otherwise, the return value is NULL, indicating the DSCancelWaitBuffer call failed.

Comments

If a wait-for-buffer request was not pending for the specified buffer number, the DSCancelWaitBuffer call still returns TRUE.

If the wait-for-buffer request was established using interrupt-driven notification (DS_WFB_POLLED for uNotifyType parameter in DSWaitForBuffer), the card-to-host server interrupt is disabled.

C/C++ Example

See Also

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSCancelWaitFlag

Related Users Guide Topics

Buffer Ready Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSCancelWaitFlag

UINT DSCancelWaitFlag (hCard, wFlagNum)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
WORD  wFlagNum    // flag number
```

The DSCancelWaitFlag function cancels a pending DSP Engine wait-for-semaphore request with the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
wFlagNum	Number of flag to cancel wait request

Returns

The return value is TRUE if the wait-for-semaphore request was canceled successfully. Otherwise, the return value is NULL, indicating the DSCancelWaitFlag call failed.

Comments

If a wait-for-semaphore request was not pending for the specified flag number, the DSCancelWaitFlag call still returns TRUE.

If the wait-for-semaphore request was established using interrupt-driven notification (DS_WFB_POLLED for uNotifyType parameter in DSWaitForFlag), the card-to-host server interrupt is disabled.

C/C++ Example

See Also

DSWaitForFlag

WM_DSPENGINE_FLGRDY

DSCancelWaitBuffer

Related Users Guide Topics

Semaphore Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSDisableCard

UINT DSDisableCard (hCard)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
```

The DSDisableCard function disables the specified multicore CPU or DSP/analog hardware, placing all cores in reset and disabling other oncard resources which may be active.

 Copyright © Signalogic 1994-2013

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is non-zero if the multicore CPU or DSP/analog hardware is disabled successfully. Otherwise, the return value is NULL, indicating the DSDisableCard call failed.

Comments

DSDisableCard is the antithesis of DSInitCard. It places all devices on the multicore CPU or DSP/analog hardware in reset, and disables other resources that may have been initialized in the course of other Hardware Library calls and/or subsequent device-executable code operations. Examples include disabling host server memory-mapped areas (if any), turning off sampling rate clock generators, disabling output drivers, etc.

After a DSDisableCard call, a DSResetCard call should be invoked prior to making further Hardware Library calls to the multicore CPU or DSP/analog hardware.

NOTE: DSDisableCard should not be called unless a successful DSInitCard function has been completed previously.

C/C++ Example

```
DisableCard(hCard);  // shut-down card, including analog I/O
```

See Also

DSInitCard

DSResetCard

DSAssignCard

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSDSPToIEEE

UINT DSDSPToIEEE (hCard,
uIEEEDataSize,
pSrcMem,
pDstMem,
dwCount)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uIEEEDataSize	// size of each IEEE floating-point data element
PMEM pSrcMem	// pointer to source buffer containing 32-bit floating-point multicore CPU or DSP data
PMEM pDstMem	// pointer to destination data buffer which will contain IEEE floating-point data
DWORD dwCount	// number of data elements to convert

The DSDSPToIEEE function converts 32-bit floating-point data in format used by multicore CPU or DSP devices associated with hCard to IEEE 32-bit floating-point data. Both source and destination data buffers are expected to be host server memory defined by the calling application.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uIEEEDataSize	Size of individual IEEE floating-point data elements, as defined below:

Value	Meaning
DS_DTI_IEEESIZE32	32-bit IEEE data
DS_DTI_IEEESIZE64	64-bit IEEE data

pSrcMem	Address in host memory of source buffer containing multicore CPU or DSP 32-bit floating-point data
pDstMem	Address in host memory of destination buffer which will contain IEEE 32-bit floating-point data
dwCount	Number of data elements to convert

Returns

The return value is a TRUE if the conversion was completed successfully. Otherwise, the return value is NULL, indicating the DSDSPToIEEE call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The purpose of the DSDSPToIEEE and DSIEEEToDSP functions is to convert calling application floating-point data to and from native device floating-point data. These functions may be useful before and after DSReadMem and DSWriteMem functions.

The DSDSPToIEEE and DSIEEEToDSP functions determine the multicore CPU or DSP device type from the driver ID value associated with hCard. This value is established from the Driver ID field in the Hardware Setup File when a DSAssignCard call is made. Currently, for ***fixed-point multicore CPU or DSP devices,*** the DSDSPToIEEE function treats

device data as if it were in signed fractional form, which means that only IEEE data values in the range of ±1 will be generated and stored.

C/C++ Example

```
DWORD n, dwCount = 1024;
float dbuf[dwCount];

for n = 0; n<dwCount-1; n++) dbuf[n] = sin(2*pi*n*1000/10000);  // some IEEE data

// convert data to native floating-point multicore CPU or DSP device format

DSIEEEToDSP(hCard, DS_DTI_IEEESIZE32, &dbuf, &dbuf, dwCount);

// transfer converted data to memory on multicore CPU or DSP card

 DSWriteMem(hCard, DS_RM_LINEAR_DATA, 0x1000, DSGetWordLength(hCard), &dbuf, dwCount);
```

Notes about the above example

1) IEEE data is converted in-place in dbuf, which means trying to use this data again in the host server program would produce incorrect results

2) The device memory address of 0x1000 was chosen arbitrarily; depending on the type of multicore CPU or DSP devices referenced by hCard, this location could be in onchip SRAM or offchip SRAM

3) The DSGetWordLength function is a convenient method to determine the natural word-size for a given card; in the above example, it is assumed the result will be 32-bits, because of the previous use of the DS_DTI_IEEESIZE32 value

See Also

DSIEEEToDSP

DSReadMem, DSReadMemEx

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

DSFreeCard

UINT DSFreeCard (hCard)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
```

The DSFreeCard function unallocates and frees the specified card handle.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is non-zero if the multicore CPU or DSP/analog hardware handle is freed successfully. Otherwise, the return value is NULL, indicating the DSFreeCard call failed.

Comments

Further Hardware Library calls using a handle previously freed by DSFreeCard will return NULL.

DSFreeCard should be invoked for any active multicore CPU or DSP/analog hardware handles before calling applications which are using Hardware Library terminate. It may also be necessary to choose a handle to free if a recent call to DSAssignCard indicates that all handles are allocated. In this case, the DSAssignCard result is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines return the DSALLCARDHANDLESALLOCATED error condition.

It is always a good idea to unallocate multicore CPU or DSP/analog hardware handles as soon as they are no longer needed. This is because there may be multiple applications using Hardware Manager, requiring more handles at one time than is readily apparent.

C/C++ Example

See Also

DSAssignCard

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSGenerateWvfrmFile

UINT DSGenerateWvfrmFile (hCard, pConversionInfo)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
PCONVERSIONINFO pConversionInfo	// pointer to CONVERSIONINFO struct containing waveform file playback or signal generation information

The DSGenerateWvfrmFile function performs a waveform playback or continuous signal generation operation, sending data to one or more analog output channels on the specified multicore CPU or DSP/analog hardware continuously from waveform file.

Parameter	Description
hCard	Handle to multicore CPU or DSP/analog hardware previously obtained from DSAssignCard call
pConversionInfo	Pointer to CONVERSIONINFO structure containing information about the waveform file playback or continuous signal generation operation to be performed

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the operation failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Because this is an asynchronous function, calling applications are notified of any problems encountered (waveform file errors, compile errors, etc.) via a callback function. For Linux applications, notification is sent via a signal to a function previously registered with the DSRegisterCallbackFunc API. For Windows applications, notification is sent via a

WM_DSPENGINE_PROGSTATUS message, and application code must have a window message handler set up to receive such PROGSTATUS messages.

C/C++ Example

See Also

DSAcquireWvfrmFile

WM_DSPENGINE_PROGSTATUS

DSAssignCard

DSExecEngineFunction

Related Users Guide Topics

section 4.5, Acquiring Data Using DirectCore Software

Program Status Messages from DSP Engines, in section 7.4, Registering Callback Messages from DSP Engines

section 4.1.1, Engine and Card Handles in DirectCore Software

section 7.3, Executing Functions in DSP Engines

DSGetCardBaseAddr

WORD DSGetCardBaseAddr (hCard, uAddrType)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uAddrType	// type of base address to retrieve

The DSGetCardBaseAddr function retrieves the current I/O or memory base address for the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uAddrType	Type of base address to retrieve; can be one of the values defined below:

Value	Meaning
DS_GCBA_IOADDR	retrieve I/O base address
DS_GCBA_MEMADDR	retrieve memory base address

Returns

The return value is the current I/O or memory base address for the multicore CPU or DSP/analog hardware specified by hCard. Otherwise, the return value is NULL, indicating the DSGetCardBaseAddr call failed.

Comments

The values returned by DSGetCardBaseAddr are either initialized in the original DSAssignCard call which created hCard or set subsequently using a DSSetCardBaseAddr call.

Note that if either the uIOBaseAddr or uMemBaseAddr parameters in the original DSAssignCard call were NULL, then, as a result of the DSGetCardBaseAddr call, these values are initialized to the current I/O Base Address and Memory Base Address values in the Hardware Setup File.

C/C++ Example

See Also

DSSetCardBaseAddr

DSAssignCard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSGetCardBusType

UINT DSGetCardBusType (hCard)

```
HCARD hCard    // multicore CPU or DSP/analog hardware handle
```

The DSGetCardBusType function retrieves the current bus type for the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is the current bus type for the multicore CPU or DSP/analog hardware specified by hCard, shown by the values defined below. Otherwise, the return value is NULL, indicating the DSGetCardBusType call failed.

Value	Meaning
DS_AC_PCXT	PC-XT bus type (deprecated)
DS_AC_PCAT	PC-AT bus type (deprecated)
DS_AC_ENABLE80X86INST	Enable 80x86 instruction set operations (deprecated)
DS_AC_PCIEX1	PCIe bus type, 1 lane
DS_AC_PCIEX4	PCIe bus type, 4 lanes
DS_AC_PCIEX8	PCIe bus type, 8 lanes
DS_AC_PCIEX16	PCIe bus type, 16 lanes

Comments

Depending on the speed of the host server, and the type of multicore CPU or DSP/analog hardware, there may be cases where specifying a different bus type can make the difference between correct operation of the hardware, or improve performance. For example, some legacy types of multicore CPU or DSP/analog hardware do not operate reliably when 80x86 type "string

move" instructions are used, in which case the DS_AC_ENABLE80X86INST bus type attribute shown above should be avoided. For complete bus type information for specific hardware, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide.

The value returned by DSGetCardBusType is either that initialized in the original DSAssignCard call which created hCard or set subsequently using a DSSetCardBusType call.

Note that if the uBusType parameter in the original DSAssignCard call was NULL, then as a result of the DSGetCardBusType call, this value is initialized to the current Bus Type value in the Hardware Setup File.

C/C++ Example

See Also

DSSetCardBusType

DSAssignCard

Related Users Guide Topics

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSGetCardClass

WORD DSGetCardClass (hCard)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
```

The DSGetCardClass function retrieves a classification value that can be given to programs running on the multicore CPU or DSP/analog hardware specified by hCard, and used by those programs to enumerate specific hardware capabilities and features.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is a classification value for the specified multicore CPU or DSP/analog hardware. Since classification values may be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked after this call (see C/C++ source code example below).

Comments

The DSGetCardClass function returns a hardware classification value suitable for use by RTAF software. This value is maintained by RTAF software because device types may support multiple forms of multicore CPU or DSP/analog hardware, each with different configuration and analog I/O driver characteristics. Classification values should be sent to executing multicore CPU or DSP code prior to run-time initialization.

The result of the DSGetCardClass function can be sent directly to the RTAF software DSPROP_CARDCLASS and DSPROP_CARDSUBCLASS properties (see source code example below).

For a list of classification values, see section 3.2, Card Designator, Driver ID, and Classification Table.

A successful DSInitCard function must have been completed beforehand.

C/C++ Example

See Also

DSCalcSampFreq

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSSetProperty

DSPROP_CARDCLASS

DSPROP_CARDSUBCLASS

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSGetCardInfo

UINT DSGetCardInfo (hCard, uInfoType)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
UINT   uInfoType  // type of information to retrieve
```

The DSGetCardInfo function retrieves internal Hardware Library information for the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uInfoType	Type of information to retrieve, as indicated by one of the values defined below:

Value	Meaning
DS_GCI_ENGINEHANDLE	Engine handle used in original DSAssignCard call (if applicable, see comments in DSAssignCard call description)
DS_GCI_DRIVERID	Driver ID value
DS_GCI_CARDTYPE	Card type (currently same as Driver ID)
DS_GCI_CALLSALLOWED	Call groups allowed flag
DS_GCI_HWMGRENTRYINDEX	Hardware Setup File index (only used in Windows applications)
DS_GCI_WORDLENGTH	Natural CPU and memory word length of devices on the card (return value given in number of bits)
DS_GCI_GETCARDFIRST	Used to enumerate other info types, such as Module

DS_GCI_GETCARDNEXT	ID, mfg card handle, etc.
DS_GCI_GETCARDLAST	
DS_GCI_MODULEID	Module ID. Used in conjunction with DS_GCI_CARDFIRST, DS_GCI_CARDNEXT, and DS_GCI_CARDLAST to return IDs of daughtermodules on the card (e.g. analog I/O or other type of modules)
DS_GCI_MFGDRIVERHANDLE	Used internally
DS_GCI_MFGCARDHANDLE	Used internally
DS_GCI_NUMDEVICES	Number of devices on the card specified by hCard
DS_GCI_NUMCORESPERDEVICE	Number of cores per device on the card specified by hCard

Returns

The return value is the specified type of information for the multicore CPU or DSP/analog hardware indicated by hCard. Since all types of return values except for engine handles could legitimately be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked after this call (see C/C++ source code example below).

Comments

For more information about driver IDs and the call groups allowed flag, see Signalogic Software Users Guide section 8.3.2, Hardware Setup File Access; see also section 5.1.1 HWLISTENTRY Structure, which describes the structure used to contain individual Hardware Setup File entries.

When uInfoType is DS_GCI_HWMGRENTRYINDEX, the return value can be used directly in the DSGetHWMgrEntry and DSSetHWMgrEntry function calls in Hardware Manager.

C/C++ Example

See Also

DSAssignCard

DSGetHWMgrEntry

HWLISTENTRY

DSCalcSampFreq

DSGetMemSize

DSGetMemArch

DSGetWordLength

DSGetCardClass

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

HWLISTENTRY Structure, in section 8.3.2, Hardware Setup File Access

section 4.3.3, Hardware Interrogation

DSGetCoreList

QWORD DSGetCoreList (hCard)

HCARD hCard // multicore CPU or DSP/analog hardware handle

The DSGetCoreList function returns a list of currently active cores.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value contains the currently selected core list. Zero is a valid return value, indicating that no cores are currently selected.

Comments

For more information about how the currently selected core list works, see section 2.3 below, About Core Lists.

Note that the DSGetCoreList function does not actually access hardware. It only accesses an internal memory database used to control and monitor core usage.

NOTE: DSGetCoreList should not be called unless a successful DSInitCard function has been completed previously.

C/C++ Example

```
if (DSGetCoreList(hCard)) {  // get currently selected cores – note that printf won't occur if at least one core is not currently selected

    printf("call to DSGetCoreList returns0x %lx \n");
}
```

See Also

DSSetCoreList

DSInitCard, DSInitCores

DSLoadFileCard, DSLoadFileCores

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore software

 Copyright © Signalogic 1994-2013

DSGetProperty

DWORD DSGetProperty (hCard, dwProperty)

```
HCARD  hCard        // multicore CPU or DSP/analog hardware handle
DWORD  dwProperty   // property name
```

The DSGetProperty function reads a data element from the specified RTAF property in memory on the multicore CPU or DSP/analog hardware indicated by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
dwProperty	RTAF property to read; for a list of possible properties, see section 2.3, RTAF Properties

Returns

The return value is the RTAF software property if read successfully. Since property values can be zero, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked (see C/C++ source code example below).

Comments

The DSGetProperty and DSSetProperty functions are specific functions designed to transfer data between calling applications and RTAF software which has been loaded or is executing on multicore CPU or DSP/analog hardware. RTAF software supports a wide variety of multicore CPU or DSP/analog hardware types, and contains several built-in "operating modes" which can be controlled from host server applications.

For general information, see Signalogic Software Users Guide section 5.0.2, RTAF Software.

Note that all RTAF properties are maintained in multicore CPU or DSP program memory. Although the dwProperty parameter could be any value, it should be one of the documented RTAF properties. For a list of RTAF properties, see section 2.3, RTAF Properties. In cases where user-defined dwProperty values are used, the actual program memory location accessed can be calculated as:

```
dwPhysicalAddr = dwProperty – DSPROP_CARDCLASS + PropertyStartAddr
```

DSPROP_CARDCLASS is a RTAF property, as explained above (it is the first entry in the property list maintained by RTAF software). In RTAF software, PropertyStartAddr is specific for each DSP device family. Note also that even though DSGetProperty allows for 32-bit values to be read, some DSP devices have a 16-bit or 24-bit natural word length. In these cases, only the least significant word length bits are read. The DSGetWordLength function can be used to determine word length for specified types of multicore CPU or DSP/analog hardware. Below is a table with common PropertyStartAddr and word length values.

Multicore CPU or DSP Family	PropertyStartAddr	Word Length
ADSP-21xx	0x1c	16
DSP32C	0x40	32
DSP3210	0x40	32
DSP5600x	0x40	24
DSP9600x	0x100	32
TMS320C25	0x42f	16

TMS320C3x	0x400	32
TMS320C4x	0x400	32
TMS320C54x	0x42f	16
TMS320C55x	0x2000	16
TMS320C64x	0x2000	32
TMS320C66x	0x2000	32

All values for PropertyStartAddr listed above are given as if DSP device program memory is accessible only as contiguous locations each of word length bits. Values marked with † indicate that the DSP device may actually be byte- or word-addressable, but only the word length value shown is used for DSGetProperty and DSSetProperty addressing.

C/C++ Example

Below is an example of reading a RTAF property:

```
DWORD dwTimDataBaseAddr;

dwTimDataBaseAddr = DSGetProperty(hCard, DSPROP_TIMDATAADDR);

if (!dwTimDataBaseAddr) {

    HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE);
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

    if (nErrorType != NULL) {

        // do error processing...
    }
}
```

The example above reads the default time data base address used by native RTAF functions and instruments. This example assumes one or more DSLoadFileCard or DSLoadFileCores calls has previously been invoked, which would cause RTAF properties to be initialized, depending on the program file that was downloaded.

See Also

DSSetProperty

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

DSGetWordLength

DSDSPToIEEE, DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

section 5.0.1, RTAF Software Interface Documentation

section 5.0.2, RTAF Software Interfaces

section 5.0.3, RTAF for Hypersignal in Engine Mode

DSGetSymbolAddr

UINT DSGetSymbolAddr (hCard,
pszCodeFile,
pszSymbolName)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
LPCSTR pszCodeFile	// pointer to string containing executable multicore CPU or DSP code filename
LPCSTR pszSymbolName	// pointer to string containing symbol name

The DSGetSymbolAddr function looks up a symbol name from the debug information contained with the specified executable code file.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
pszCodeFile	Pointer to null-terminated string containing pathname of the executable code file to download. If this parameter is NULL, then the executable code file previously downloaded by the most recent DSLoadFileCard or DSLoadFileCores API call is used
pszSymbolName	Pointer to null-terminated string containing the name of the symbol to look up

Returns

Physical memory address of the specified symbol. Note that a symbol address may legitimately be zero (although that's unlikely as typically the zero address in device memory is reserved for a Reset Vector or other device-specific purpose), so it's advisable to check for an API error using the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Usually one or more successful DSLoadFileCard or DSLoadFileCores calls should have been completed before a DSGetSymbolAddr call is attempted. In all cases, a successful DSAssignCard call must have been completed beforehand.

If the pszCodeFile parameter is NULL, the DSGetSymbolAddr call will assume the current "active" executable program code file, corresponding to previous DSLoadFileCard or DSLoadFileCores calls.

Device executable code files should be produced directly by the multicore CPU or DSP device manufacturer's build tools. Typically, these files are COFF (Common Object File Format) or ELF (Executable Linkable Format) format.

Note that symbol names are case-sensitive and should ***exactly*** match their declaration in device C/C++ source code and build tools. Note that for COFF format build tools, a leading underscore may be needed. In all cases, it's advisable to carefully check the .map file (or other linker output / log file) produced by the device build tools, in order to determine precise symbol names to use in DSGetSymbolAddr calls.

C/C++ Example

```
UINT uTestMode = 1;
DWORD dwPhysAddr_uTestMode;

// download specified program (code) file (core 0 only)

if (DSLoadFileCores(hCard, "programcode.out", 0x01)) {

    // perform symbol lookup – note that symbol names are case-sensitive and must be specified precisely
```

```
    dwPhysAddr_uTestMode = DSGetSymbolAddr(hCard, NULL, "_uTestMode");  // leading underscore typically required for COFF
                                                                        // format code files

    if (!dwPhysAddr_uTestMode)
     dwPhysAddr_uTestMode = DSGetSymbolAddr(hCard, NULL, "uTestMode");  // no leading underscore required for ELF
                                                                        // format code files

    if (dwPhysAddr_uTestMode) {  // check for valid symbol physical memory address

      // Set value in multicore CPU or DSP memory

        DSWriteMem(hCard, DS_RM_LINEAR_DATA, dwPhysAddr_uTestMode, DS_RM_SIZE32, &uTestMode, 1);
    }
    else printf("DSGetSymbolAddr returned zero – this is likely an error.  Suggest checking the symbol name for exact spelling\n");
}
else printf("call to DSLoadFileCores Failed\n");
```

The source code example above first tries COFF format for the program file format, and if that fails, then tries ELF format. This is just for illustration purposes only, as normally it's known ahead of time what program file format is expected.

See Also

DSLoadFileCard, DSLoadFileCores

DSAssignCard

DSInitCard

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetMemArch

UINT DSGetMemArch (hCard)

```
HCARD  hCard     // multicore CPU or DSP/analog hardware handle
```

The DSGetMemArch function retrieves the memory architecture for the multicore CPU or DSP/analog hardware specified by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

If the DSGetMemArch function is successful, the return value is the type of memory architecture used on the specified multicore CPU or DSP/analog hardware, as defined by the values below. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Value	Meaning
DS_GMA_LINEAR	Memory architecture is linear, with program and data memory in the same address space

DS_GMA_HARVARD	Memory architecture is modified Harvard, with program and data memory in separate address spaces
DS_GMA_VECTOR	Memory architecture is modified Harvard, except data memory is further subdivided into two (2) vector spaces

Comments

The DSGetMemArch function returns a memory architecture type value suitable for use in DSReadMem and DSWriteMem functions (see source code example below).

Typical examples of multicore CPU or DSP devices which have linear memory architecture are Texas Instruments C64x and C66x series, AT&T DSP32C and DSP3210, and Texas Instruments TMS320C3x and TMS320C4x. Typical examples of devices which have a modified Harvard memory architecture are Texas Instruments TMS320C54x. Typical examples of devices which have vector memory architecture are Freescale DSP5600x.

A successful DSInitCard function must have been completed beforehand.

C/C++ Example

See Also

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

DSCalcSampFreq

DSGetMemSize

DSGetWordLength

DSGetCardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

DSGetMemSize

DWORD DSGetMemSize (hCard)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
DWORD  dwCoreNum  // value specifying which core's data memory to size
```

The DSGetMemSize function retrieves the maximum data memory size for the core specified by dwCoreNum on the multicore CPU or DSP/analog hardware specified by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
dwCoreNum	Value containing core number

Returns

If the DSGetMemSize function is successful, the return value is the maximum data memory size available for use with the specified core. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Comments

The DSGetMemSize function will return NULL if the dwCoreNum parameter is outside the range of possible cores on the multicore CPU or DSP/analog hardware, or has more than one bit set (causing confusion as to which exact core has been specified).

The DSGetMemSize function returns the maximum amount of data memory detected for the core specified by dwCoreNum. For most hardware types, the return value is in units of the native word length of the device. Some examples include 32-bit words for TMS320C64x, TMS320C66x, TMS320C3x and TMS320C4x devices, 24-bit words for DSP5600x devices, and 16-bit words for ADSP-21xx and TMS320C5x devices. However, there are currently two (2) exceptions. The DSGetMemSize return value is in 16-bit words for DSP32C and DSP3210 devices, even though these devices are 32-bit floating-point. This is because these devices have true multiwidth access to onchip and external memory, and some DSP Engine drivers take advantage of this feature to store data (especially analog I/O data) in compact (16-bit integer) form when possible.

In cases where the multicore CPU or DSP/analog hardware has separate program and data memory areas, the DSGetMemSize function does not measure program memory.

In some cases, the return value may either be somewhat less than or exceed the amount of installed oncard, external data memory. This may be because external memory addressing is contiguous with onchip memory, or overlaps onchip memory (creating an unusable region).

A successful DSInitCard function must have been completed beforehand.

C/C++ Example

See Also

DSGetMemArch

DSGetWordLength

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

DSCalcSampFreq

DSGetCardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

DSGetWordLength

WORD DSGetWordLength (hCard)

```
HCARD  hCard    // multicore CPU or DSP/analog hardware handle
```

The DSGetWordLength function retrieves the natural word length of the device(s) on the multicore CPU or DSP/analog hardware specified by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

If the DSGetWordLength function is successful, the return value is the natural word length, as defined in the table below, of the device(s) used on the multicore CPU or DSP/analog hardware. Otherwise, the return value is NULL and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked for error conditions.

Value	Meaning
DS_RM_SIZE16	Device is 16-bit fixed-point
DS_RM_SIZE24	Device is 24-bit fixed-point
DS_RM_SIZE32	Device is 32-bit floating-point

Comments

Some typical examples of DSGetWordLength return values are shown in the table below.

Device Type	Native Word Length
AT&T DSP32C, DSP3210	DS_GMA_SIZE32
Analog Devices ADSP-21xx	DS_GMA_SIZE16
Analog Devices ADSP-21xxx	DS_GMA_SIZE32
Freescale DSP5600x	DS_GMA_SIZE24
Freescale DSP9600x	DS_GMA_SIZE32
Texas Instruments TMS320C25	DS_GMA_SIZE16
Texas Instruments TMS320C3x, TMS320C4x	DS_GMA_SIZE32
Texas Instruments TMS320C54x, TMS320C55x	DS_GMA_SIZE16
Texas Instruments TMS320C64x, TMS320C66x	DS_GMA_SIZE32

Note that this function does not actually "touch" hardware. The Driver ID field in the Hardware Setup File is used to determine the device type.

C/C++ Example

```
wWordLength = DSGetWordLength(hCard);

if (wWordLength == NULL) {

    hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE);
    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    // do error processing....
}
```

See Also

DSGetMemSize

DSGetMemArch

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

DSCalcSampFreq

DSGetCardClass

Related Users Guide Topics

section 4.3.3, Hardware Interrogation

DSHoldCard

UINT DSHoldCard (hCard)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
```

The DSHoldCard function places all cores on the multicore CPU or DSP/analog hardware specified by hCard into the hold state, if the hardware and multicore CPU or DSP devices support this state.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is non-zero if all cores were placed in the hold state successfully, or if the multicore CPU or DSP devices do not support this mode. Otherwise, the return value is NULL, indicating the DSHoldCard call failed.

Comments

Upon successful return, all devices on the multicore CPU or DSP/analog hardware will be in the hold state--as long as a hold state is supported by the multicore CPU or DSP/analog hardware and the device itself. For example, some older Texas Instruments TMS320 series devices can be placed in hold state, and only some multicore CPU or DSP/analog hardware types with these devices allow hold state to be enabled from the host server. Other types of multicore CPU or DSP devices do not offer an externally-selectable hold state.

All devices are held simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if only certain cores need to be held, use the DSHoldCores function.

Note that it varies between hardware types whether data transfer operations such as DSReadMem and DSWriteMem may be executed while devices are in the hold state. See documentation for the multicore CPU or DSP/analog hardware type for specific information.

C/C++ Example

See Also

DSHoldCores

DSResetCard, DSResetCores, DSResetDevices

DSRunCard, DSRunCores, DSRunDevices

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

 Copyright © Signalogic 1994-2013

DSHoldCores

UINT DSHoldCores (hCard,
qwCoreList)

HCARD hCard // multicore CPU or DSP/analog hardware handle
QWORD qwCoreList // value specifying which cores should be held

The DSHoldCores function places one or more cores on the multicore CPU or DSP/analog hardware specified by hCard into the hold state, if the hardware and its multicore CPU or DSP devices support this state. The qwCoreList parameter is used to indicate which cores should be reset.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
qwCoreList	Value containing core list; each bit represents a core, starting with bit 0 for core 0; a 1 causes the core to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the cores specified by qwCoreList were held successfully, or if the multicore CPU or DSP devices do not support this mode. Otherwise, the return value is NULL, indicating the DSHoldCores call failed.

Comments

The DSHoldCores function will return NULL if the qwCoreList parameter indicates cores outside the range of physical cores on the multicore CPU or DSP/analog hardware, or if the qwCoreList parameter is zero (indicating no cores).

Upon successful return, devices on the multicore CPU or DSP/analog hardware specified by qwCoreList will be in the hold state--as long as a hold state is supported by the multicore CPU or DSP/analog hardware and its devices themselves. For example, most Texas Instruments TMS320 series devices can be placed in hold state; only some of the multicore CPU or DSP/analog hardware types with these cores allow hold state to be enabled from host software. Other types of multicore CPU or DSP devices do not offer an externally-selectable hold state.

DSP devices specified by qwCoreList are held simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if all cores need to be held simultaneously, it is simpler to use the DSHoldCard function.

Note that it varies between hardware types whether data transfer operations such as DSReadMem and DSWriteMem may be executed while devices are in the hold state. See documentation for the multicore CPU or DSP/analog hardware type for specific information.

C/C++ Example

See Also

DSHoldCard

DSResetCard, DSResetCores, DSResetDevices

DSRunCores, DSRunCard, DSRunDevices

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

 Copyrght © Signalogic 1994-2013

DSInitCard

UINT DSInitCard (hCard)

```
HCARD hCard    // multicore CPU or DSP/analog hardware handle
```

The DSInitCard function checks for the presence, and if found, initializes the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is non-zero if the multicore CPU or DSP/analog hardware is located and initialized successfully. Otherwise, the return value is NULL, indicating the DSInitCard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Initialization of multicore CPU or DSP/analog hardware is straightforward, and occurs in the following steps:

1. A short register or oncard memory check is made to verify that the hardware is installed correctly.
2. Any multicore CPU or DSP devices on the hardware are placed in reset state.

For those types of hardware which require it, a resident "host interface" code segment may be downloaded and allowed to run. This is necessary in cases where the design of the hardware and/or multicore CPU or DSP devices do not allow oncard memory to be accessed while the DSP device is held in reset.

Note that in all cases, if the DSInitCard function is successful, further operations such as DSReadMem and DSWriteMem can be executed at any time. *If the function return is not successful, then further operations will produce undefined results*. The one exception to this is attempting additional DSInitCard calls after using DSSetCardBaseAddr to alter the I/O Base Address and/or Memory Base Address values for the hardware.

C/C++ Example

See Also

DSAssignCard

DSLoadFileCard, DSLoadFileCores

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

 Copyright © Signalogic 1994-2013

DSIEEEToDSP

UINT DSIEEEToDSP (hCard,
uIEEEDataSize,
pSrcMem,
pDstMem,
dwCount)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uIEEEDataSize	// size of each IEEE floating-point data element
PMEM pSrcMem	// pointer to source buffer containing IEEE floating-point data
PMEM pDstMem	// pointer to destination buffer which will contain 32-bit floating-point multicore CPU or DSP data
DWORD dwCount	// number of data elements to convert

The DSIEEEToDSP function converts IEEE 32-bit floating-point data to 32-bit floating-point data in format corresponding to multicore CPU or DSP devices associated with hCard. Both source and destination data buffers are expected to be in host server memory defined by the calling application.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uIEEEDataSize	Individual size of IEEE floating-point data elements, as defined by values below:

Value	Meaning
DS_DTI_IEEESIZE32	32-bit IEEE data
DS_DTI_IEEESIZE64	64-bit IEEE data

pSrcMem	Address in host memory of source buffer containing IEEE floating-point data
pDstMem	Address in host memory of destination buffer which will contain multicore CPU or DSP 32-bit floating-point data
dwCount	Number of data elements to convert

Returns

The return value is a TRUE if the conversion was completed successfully. Otherwise, the return value is NULL, indicating the DSIEEEToDSP call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The purpose of the DSDSPToIEEE and DSIEEEToDSP functions is to convert calling application floating-point data to and from device floating-point data. These functions are most useful before and after DSReadMem and DSWriteMem functions.

The DSDSPToIEEE and DSIEEEToDSP functions determine the device type from the driver ID value associated with hCard. This value is established from the Driver ID field in the Hardware Setup File when a DSAssignCard call is made. Currently, for ***fixed-point multicore CPU or DSP devices,*** the DSIEEEToDSP function attempts to convert IEEE data to signed fractional notation, which means that IEEE data values outside of the range ±1 will "wrap around".

C/C++ Example

```
DWORD n, dwCount = 1024;
float dbuf[dwCount];

// get native floating-point data from memory on multicore CPU or DSP card

 DSReadMem(hCard, DS_RM_LINEAR_DATA, 0x1000, DSGetWordLength(hCard), &dbuf, dwCount);

// convert data to IEEE floating-point format

DSDSPToIEEE(hCard, DS_DTI_IEEESIZE32, &dbuf, &dbuf, dwCount);
```

Notes about the above source code example:

1) Native multicore CPU or DSP format data is converted in-place in dbuf, which allows the data to be used subsequently in the host server program by referencing dbuf; attempting to perform floating-point operations on dbuf data after the DSReadMem function but before the DSDSPToIEEE function would produce incorrect results

2) The device memory address of 0x1000 was chosen arbitrarily; depending on the type of DSP/acquisition card referenced by hCard, this location could be in onchip SRAM or offchip SRAM

3) The DSGetWordLength function is a convenient method to determine the natural word-size for a given card; in the above example, it is assumed the result will be 32-bits, because of the DS_DTI_IEEESIZE32 value used in the conversion

See Also

DSDSPToIEEE

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

DSLoadFileCard

UINT DSLoadFileCard (hCard, pszCodeFile)

```
HCARD  hCard        // multicore CPU or DSP/analog hardware handle
LPCSTR  pszCodeFile // pointer to executable multicore CPU or DSP code filename string
```

The DSLoadFileCard function downloads an executable code file to program memory on the specified multicore CPU or DSP/analog hardware. If the hardware contains more than one core with separate program memory, all cores receive the same executable code file. If different executable files should be downloaded to different cores, then it's recommended to use the DSLoadFileCores API.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
pszCodeFile	Pointer to null-terminated string containing pathname of the executable multicore CPU or DSP code file to download

Returns

The return value is TRUE if the executable code file is downloaded successfully. Otherwise, the return value is NULL, indicating the DSLoadFileCard call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Successful DSAssignCard and DSInitCard calls must have been completed before DSLoadFileCard calls are attempted.

If the pszCodeFile parameter is NULL, the DSLoadFileCard call will download the current Program File entry in the Hardware Setup File for the corresponding multicore CPU or DSP/analog hardware type. This value can be retrieved independently by a call to DSGetHWMgrEntry using a Hardware Setup File index. There are two ways to get this index:

1. call DSGetHWMgrEntryIndex with the original card designator string used in DSAssignCard
2. call DSGetCardInfo with the hCard parameter (same as above) and DS_GCI_HWMGRENTRYINDEX as input

Using the resulting index in a DSGetHWMgrEntry call will then retrieve a HWLISTENTRY structure, in which the pszCodeFile element is the current executable code file.

DSP executable code files should be in the format produced by the device manufacturer's build tools. Typically, these are COFF (Common Object File Format) or ELF (Executable and Linkable Format) files.

NOTE: All devices on the multicore CPU or DSP/analog hardware should be in reset state before the DSLoadFileCard call is invoked. Reset state is in effect after DSInitCard, DSResetCard, DSResetCores, and DSDisableCard calls. Otherwise, results are undefined.

C/C++ Example

```
// load specified program (code) file (if card is multicore, then all cores are loaded with same file; see DSLoadFileCores below for
// core-specific program file download)

if (!DSLoadFileCard(hCard, “programcode.out”)) {

    // try default code file instead

    if (!DSLoadFileCard(hCard, NULL)) {

        MessageBox(NULL, " ", "Problem loading program file", MB_OK);
    }
}
```

See Also

DSLoadFileCores

DSAssignCard

DSInitCard

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSLoadFileCores

UINT DSLoadFileCores (hCard,
pszCodeFile,
qwCoreList)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
LPCSTR pszCodeFile	// pointer to executable multicore CPU or DSP code filename string
QWORD qwCoreList	// value specifying which cores should receive the program code file

The DSLoadFileCores function downloads an executable code file to program memory for one or more cores on the specified multicore CPU or DSP/analog hardware. The qwCoreList parameter is used to indicate which cores should receive the executable program code file.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
pszCodeFile	Pointer to null-terminated string containing pathname of the executable code file to download
qwCoreList	Value containing core list; each bit represents a core, starting with bit 0 for core 0; a 1 enables downloading, a 0 disables

Returns

The return value is TRUE if the executable code file is downloaded successfully. Otherwise, the return value is NULL, indicating the DSLoadFileCores call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

Successful DSAssignCard and DSInitCard calls must have been completed before DSLoadFileCores calls are attempted.

The DSLoadFileCores function will return NULL if the qwCoreList parameter indicates cores outside the range of physical cores on the multicore CPU or DSP/analog hardware, or if the qwCoreList parameter is zero (indicating no cores).

If the pszCodeFile parameter is NULL, the DSLoadFileCores call will download the current Program File entry in the Hardware Setup File for the corresponding multicore CPU or DSP/analog hardware type. This value can be retrieved independently by a call to DSGetHWMgrEntry using a Hardware Setup File index. There are two ways to get this index:

1. call DSGetHWMgrEntryIndex with the original card designator string used in DSAssignCard
2. call DSGetCardInfo with the hCard parameter (same as above) and DS_GCI_HWMGRENTRYINDEX as input

Using the resulting index in a DSGetHWMgrEntry call will then retrieve a HWLISTENTRY structure, in which the pszCodeFile element is the current executable code file.

Device executable code files should be in the format produced by the device manufacturer's build tools. Typically, these are COFF (Common Object File Format) or ELF (Executable Linkable Format) files.

For a detailed description of qwCoreList behavior, see section 2.3.2 below, API Core List Parameters.

NOTE: All devices on the multicore CPU or DSP/analog hardware should be in reset state before the DSLoadFileCard call is invoked. Reset state is in effect after DSInitCard, DSResetCard, DSResetCores, and DSDisableCard calls. Otherwise, results are undefined.

C/C++ Example

```
// load specified program (code) file (core 0 only)

if (!DSLoadFileCores(hCard, “programcode.out”, 0x01)) {

    // try default code file (cores 0, 1, and 2)

    if (!DSLoadFileCores(hCard, NULL, 0x07)) {

        MessageBox(NULL, " ", "Problem loading program file", MB_OK);
    }
}
```

See Also

DSLoadFileCard

DSAssignCard

DSInitCard

Related Users Guide Topics

section 4.1.1, Engine and Card Handles in DirectCore Software

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

Program File Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSSetProperty

UINT DSSetProperty (hCard,
dwProperty,
dwPropertyData)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
DWORD dwProperty	// property name
DWORD dwPropertyData	// data to write

The DSSetProperty function writes a data element to the specified RTAF property in program memory on the multicore CPU or DSP/analog hardware indicated by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
dwProperty	RTAF property to write; for a list of possible variables, see section 2.3, RTAF Properties
dwPropertyData	Data to write to RTAF property

Returns

The return value is a TRUE if the property was written successfully. Otherwise, the return value is NULL, indicating the DSSetProperty call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSGetProperty and DSSetProperty functions are specific functions designed to transfer data between calling applications and RTAF software which has been loaded or is executing on multicore CPU or DSP/analog hardware. RTAF software supports a wide variety of multicore CPU or DSP/analog hardware types, and contains several built-in "operating modes" which can be controlled from host server applications.

For general information, see Signalogic Software Users Guide section 5.0.2, RTAF Software.

Note that all RTAF properties are maintained in program or combined program/data memory on the multicore CPU or DSP hardware. Although the dwProperty parameter can be any value, it should be one of the documented RTAF properties. For a list of RTAF properties, see section 2.3, RTAF Properties. In cases where user-defined dwProperty values are used, the actual memory location accessed can be calculated as:

dwPhysicalAddr = dwProperty – DSPROP_CARDCLASS + PropertyStartAddr

DSPROP_CARDCLASS is a RTAF property, as explained above (it is the first entry in the property list maintained by RTAF software). In RTAF software, PropertyStartAddr is specific for each device family. Note also that even though dwPropertyData allows for 32-bit values to be written, some multicore CPU or DSP devices have a 16-bit or 24-bit natural word length. In these cases, only the least significant word length bits are written. The DSGetWordLength function can be used to determine word length for specified types of multicore CPU or DSP/analog hardware. Below is a table with common PropertyStartAddr and world length values.

Multicore CPU or DSP Family	PropertyStartAddr	Word Length
ADSP-21xx	0x1c	16
DSP32C	0x40	32
DSP3210	0x40	32
DSP5600x	0x40	24
DSP9600x	0x100	32
TMS320C25	0x42f	16
TMS320C3x	0x400†	32
TMS320C4x	0x400	32
TMS320C54x	0x42f	16
TMS320C55x	0x2000	16
TMS320C64x	0x2000	32
TMS320C66x	0x2000	32

All values for PropertyStartAddr listed above are given as if multicore CPU or DSP device program memory is accessible only as contiguous locations each of word length bits. Values marked with † indicate that the multicore CPU or DSP device may actually be byte- or word-addressable, but only the word length value shown is used for DSGetProperty and DSSetProperty addressing.

C/C++ Example

Below is a short excerpt from the dscope.cpp source code example file included with DirectCore software:

```
DWORD dwFsMode;
WORD wCardClass;
float FsActual;

// interrogate engine for card type value
wCardClass = DSGetCardClass(hCard);

// find sample rate control register value and actual rate (closest rate possible to desired rate)
// 22.05 kHz sampling rate, assume simultaneous sampling
```

```
dwFsMode = DSCalcSampFreq(hCard, 22050.0, NULL, NULL, &FsActual);

// load multicore CPU or DSP device program file (usually a COFF or ELF format file produced by the device's build tools)
if (!DSLoadFileCores(hCard, NULL, 0x01)) {  // load default file (core 0 only)

    MessageBox(NULL, " ", "Dscope Demo:  problem loading file", MB_OK);
    goto cleanup;
}

// send down RTAF properties in anticipation of using a RTAF operating mode
…
...
DSSetProperty(hCard, DSPROP_CARDCLASS, wCardClass & 0x00ff);
DSSetProperty(hCard, DSPROP_CARDSUBCLASS, wCardClass >> 8);
...
DSSetProperty(hCard, DSPROP_FSMODE, dwFsMode);
...
```

See Also

DSGetProperty

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

DSGetWordLength

DSDSPToIEEE, DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

section 5.0.1, RTAF Interface Documentation

section 5.0.2, RTAF Interfaces

section 5.0.3, RTAF for Hypersignal in Engine Mode

DSReadCardReg

WORD DSReadCardReg (hCard, wRegOffset)

```
HCARD  hCard       // multicore CPU or DSP/analog hardware handle
WORD  wRegOffset   // offset from register base address
```

The DSReadCardReg function reads a value from the specified register on the multicore CPU or DSP/analog hardware indicated by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
wRegOffset	Offset from I/O base address of multicore CPU or DSP/analog hardware

Returns

The return value is the current value of specified register on the multicore CPU or DSP/analog hardware, assuming it can be read. Since register values can legitimately be NULL, the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should always be checked (see C/C++ source code example below).

Comments

In cases where directly accessing multicore CPU or DSP/analog hardware is unavoidable, the DSReadCardReg and DSWriteCardReg functions may prove convenient.

In cases where registers are write-only, the return result is likely to be 0xffff. This may also be the case if a location being read is actually not a card register. Consult the multicore CPU or DSP/analog hardware documentation for detailed information about I/O address space registers the hardware may support.

C/C++ Example

```
wVal = DSReadCardReg(hCard, wOffset);

HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE);
int nErrorType = DSGetEngineErrorStatus(hEngine);
if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

if (nErrorType != NULL) {

    // do error processing...
}
```

See Also

DSWriteCardReg

DSSetCardBaseAddr

DSAssignCard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

 Copyright © Signalogic 1994-2013

DSReadMem

UINT DSReadMem (hCard,
uMemType,
dwDeviceMemAddr,
uBitWidth,
pMem,
dwCount)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uMemType	// multicore CPU or DSP memory type
DWORD dwDeviceMemAddr	// multicore CPU or DSP memory start address
UINT uBitWidth	// data element width in multicore CPU or DSP memory
PMEM pMem	// pointer to buffer to receive data
DWORD dwCount	// number of data elements to transfer

The DSReadMem function transfers one or more data elements from multicore CPU or DSP/analog hardware memory to host server memory defined by the calling application.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uMemType	Type of multicore CPU or DSP memory from which to transfer data elements, as defined below:

Value	Meaning
DS_RM_DIRECTACCESS	Currently deprecated; used with some legacy hardware
DS_RM_L2	Core specific L2 memory (typically onchip memory)
DS_RM_LINEAR_PROGRAM	Linear program memory
DS_RM_LINEAR_DATA	Linear data memory
DS_RM_LINEAR_DATA_RT	Linear data memory fast transfer while core is in run state (used only for some legacy DSPs only)
DS_RM_MCSM	Multicore shared memory (typically onchip memory)
DS_RM_READMULTIPLE	Controls how host memory size and format expectations when reading multiple cores simultaneously
DS_RM_EXTSDRAM	Device external SDRAM memory, for example DDR3 SDRAM
DS_RM_VECTOR_DATA_X	Vector X: data memory
DS_RM_VECTOR_DATA_Y	Vector Y: data memory

dwDeviceMemAddr	Starting address in multicore CPU or DSP data memory for transfer
uBitWidth	Data element width in multicore CPU or DSP memory for transfers, as defined below:

Value	Meaning
DS_RM_SIZE8	Device memory data element width is 1 byte
DS_RM_SIZE16	Device memory data element width is WORD
DS_RM_SIZE24	Device memory data element width is 3 bytes
DS_RM_SIZE32	Device memory data element width is DWORD or float

pMem	Pointer to calling application memory which will receive data elements
dwCount	Number of data elements to transfer

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSReadMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSReadMem and DSWriteMem functions are flexible, general-purpose functions intended to transfer data between host server memory and oncard or onchip multicore CPU or DSP/analog hardware memory. Because of the wide variety of multicore CPU or DSP/analog hardware memory types, and the flexibility of the DSReadMem and DSWriteMem APIs, there are several items to keep in mind:

1) For some types of multicore CPU or DSP/analog hardware, uBitWidth values of DS_RM_SIZE8 and DS_RM_SIZE24 are not supported. In cases where a uBitWidth value is not supported, the transfer defaults to the next higher data element size.

2) If the multicore CPU or DSP hardware supports multicore shared memory, L2 core specific memory, or external SDRAM memory, then the DS_RM_MCSM, DS_RM_L2, and DS_RM_EXTSDRAM memory type attributes can be used, and the DS_RM_LINEAR_xxx attributes can be ignored. Note that in some cases the physical device address may make an attribute superfluous; for instance, for C66x hardware if DS_RM_EXTSDRAM is combined with an external device address higher than 0x80000000, then the attribute is redundant.

3) For 32-bit multicore CPU or DSP devices which support byte- or word-addressable memory, addressing used by DSReadMem and DSWriteMem appears as if device memory is accessible as contiguous locations each of uBitWidth bits. For example, if a device supports byte-addressable memory, then using a uBitWidth value of DS_RM_SIZE16 would cause access to occur at the *byte address* of dwDeviceDataAddr∗2, and a uBitWidth value of DS_RM_SIZE32 would cause access to occur at the byte address of dwDeviceMemAddr∗4 (or the *word address* of dwDeviceMemAddr∗2). Examples of byte-addressable devices include the Texas Instruments C64x and C66x series, AT&T DSP32C and DSP3210, and Texas Instruments TMS320C32.

4) Currently, no dynamic data conversion between integer and floating-point or IEEE floating-point and native device floating-point formats is made by DSReadMem or DSWriteMem. For example, if a calling application is transferring 64-bit floating-point data to a multicore CPU or DSP program, the program may need to “know” to convert this data to its native 32-bit floating-point format before processing.

5) Currently, the DS_RM_VECTOR_DATA_X and DS_RM_VECTOR_DATA_Y uMemType attributes are only supported for Freescale DSP5600x and DSP9600x devices. DS_RM_VECTOR_DATA_X and DS_RM_LINEAR_DATA are equivalent.

C/C++ Example

The source code example below transfers 32-bit data from device data memory address 0xc0000 to a calling application data buffer created by the Windows GlobalAlloc function. Note that the actual type of data being transferred could be either 32-bit integer or 32-bit floating-point, although the example assumes the data is floating-point.

```
HGLOBAL hBuf;
float* pBuf;
```

```
WORD wBuflen;
DWORD dwDeviceMemAddr = 0xc0000;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBuflen*sizeof(float));
pBuf = (float*)GlobalLock(hBuf);

UINT uStatus = DSReadMem(hCard, DS_RM_LINEAR_DATA, dwDeviceMemAddr, DS_RM_SIZE32, pBuf, wBuflen);

if (!uStatus) {

    HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSReadMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that a 32-bit floating-point device is assumed. The above example would not be recommended for a 16-bit or 24-bit fixed-point device.

See Also

DSWriteMem, DSWriteMemEx, DSReadMemEx

DSGetProperty, DSSetProperty

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE, DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

 Copyrght © Signalogic 1994-2013

DSReadMemEx

UINT DSReadMemEx (hCard,
uMemType,
dwDeviceMemAddr,
uBitWidth,
pMem,
dwCount,
qwCoreList)

```
HCARD  hCard            // multicore CPU or DSP/analog hardware handle
UINT  uMemType          // multicore CPU or DSP memory type
DWORD  dwDeviceMemAddr  // multicore CPU or DSP memory start address
UINT  uBitWidth         // data element width in multicore CPU or DSP memory
PMEM  pMem              // pointer to buffer to receive data
DWORD  dwCount          // number of data elements to transfer
QWORD  qwCoreList       // list of cores from which to read memory
```

The DSReadMemEx function transfers one or more data elements from multicore CPU or DSP/analog hardware memory to host server memory defined by the calling application.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uMemType	Type of multicore CPU or DSP memory from which to transfer data elements, as defined below:

Value	Meaning
DS_RM_DIRECTACCESS	Currently deprecated; used with some legacy hardware
DS_RM_L2	Core specific L2 memory (typically onchip memory)
DS_RM_LINEAR_PROGRAM	Linear program memory
DS_RM_LINEAR_DATA	Linear data memory
DS_RM_LINEAR_DATA_RT	Linear data memory fast transfer while core is in run state (used only for some legacy DSPs only)
DS_RM_MCSM	Multicore shared memory (typically onchip memory)
DS_RM_READMULTIPLE	Controls how host memory size and format expectations when reading multiple cores simultaneously
DS_RM_EXTSDRAM	Device external SDRAM memory, for example DDR3 SDRAM
DS_RM_VECTOR_DATA_X	Vector X: data memory
DS_RM_VECTOR_DATA_Y	Vector Y: data memory

dwDeviceMemAddr Starting address in multicore CPU or DSP data memory for transfer

uBitWidth Data element width in multicore CPU or DSP memory for transfers, as defined below:

Value	Meaning
DS_RM_SIZE8	Device memory data element width is 1 byte
DS_RM_SIZE16	Device memory data element width is WORD
DS_RM_SIZE24	Device memory data element width is 3 bytes
DS_RM_SIZE32	Device memory data element width is DWORD or float

pMem Pointer to calling application memory which will receive data elements

dwCount Number of data elements to transfer

qwCoreList List of cores from which to read memory

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSReadMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

See the comments in the DSReadMem and DSWriteMem function descriptions.

For a detailed description of qwCoreList behavior, see section 2.3.2 below, API Core List Parameters.

If multicore cores and the DS_RM_READMULTIPLE attribute are specified and the device memory address (or memory type attribute) does not specify core-specific memory -- for example specifying onchip memory or external device memory shared by all cores – then results are undefined, although DirectCore and its driver software may attempt to recognize the redundancy and physically read data only once and then replicate (copy) in the return buffer space (this behavior is not guaranteed on all systems).

C/C++ Example

The source code example below transfers 32-bit data from data memory of eight (8) cores at address 0xc0000 to a calling application data buffer created by an operating-system dependent memory allocation function. Note that the actual type of data being transferred could be either 32-bit integer or 32-bit floating-point, although the example assumes the data is floating-point.

```
HGLOBAL hBuf;
float* pBuf;
WORD wBuflen;
DWORD dwDeviceMemAddr;
unsigned int uNumCores = 8;
DWORD dwCoreList = pow(2, uNumCores)-1;

dwDeviceMemAddr = 0xc0000;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBuflen*sizeof(float)*uNumCores);
pBuf = (float*)GlobalLock(hBuf);

UINT uStatus = DSReadMemEx(hCard, DS_RM_LINEAR_DATA, dwDeviceMemAddr, DS_RM_SIZE32, pBuf, wBuflen, dwCoreList);

if (!uStatus) {

    HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
```

```
    MessageBox(NULL, tmpstr, “DSReadMem error occurred”, MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Notes About Source Code Example #1

1) A 32-bit floating-point device is assumed. The above example could be modified for a 16-bit or 24-bit fixed-point device by appropriate use of the uBitWidth parameter.

2) dwCoreList will be zero-extended to 64 bits when passed as an argument to DSReadMemEx.

C/C++ Example 2

The source code example reads from multiple cores; note the requirement to allow enough host memory to receive multicore data correctly.

```
DWORD dwDeviceMemAddr;
UINT uNumCores = 48;
QWORD qwCoreList = pow(2, uNumCores) – 1;
DWORD dwDeviceMemAddr = 0x800;
DWORD dwVal[uNumCores];
UINT uBitWidth = DSGetWordLength(hCard);
UINT uStatus;

uStatus = DSReadMemEx(hCard, DS_RM_LINEAR_DATA | DS_RM_READMULTIPLE, dwDeviceMemAddr, uBitWidth, dwVal, 1,
qwCoreList);

if (uStatus) {

    for (i=0; i<uNumCores; i++)
      printf(“value read from core 0x%x memory = 0x%08lx\n”);
}
else {

    // do error processing…
}
```

See Also

DSReadMem, DSWriteMem, DSWriteMemEx

DSGetProperty, DSSetProperty

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE, DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

 Copyright © Signalogic 1994-2013

DSResetCard

UINT DSResetCard (hCard)

```
HCARD  hCard    // multicore CPU or DSP/analog hardware handle
```

The DSResetCard function places all cores on the multicore CPU or DSP/analog hardware specified by hCard into the reset state. Usually, this also includes a Reset of all interfaces and other circuitry on the card.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is non-zero if all cores were reset successfully. Otherwise, the return value is NULL, indicating the DSResetCard call failed.

Comments

Upon successful return, all devices on the multicore CPU or DSP/analog hardware will be in the reset state, with no code executing on devices on the card.

All multicore CPU or DSP devices are reset simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if only certain cores need to be reset, use the DSResetCores function.

The DSP Engine is responsible for accounting for necessary delays or special timing due to reset generators or other reset circuitry.

For some types of hardware, the DSResetCard call may actually return with multicore CPU or DSP devices in the run state, in which case a resident "host interface" code segment will be active on the device. This is necessary in cases where the design of the hardware and/or device does not allow oncard memory to be accessed while the device(s) are held in reset.

Note that in all cases, if the DSResetCard function is successful, further operations such as DSReadMem and DSWriteMem can be executed at any time. *If the DSResetCard function is not successful, then further operations will produce undefined results*.

NOTE: DSResetCard should not be called unless a successful DSInitCard function has been completed previously.

C/C++ Example

```
if (!DSResetCard(hCard)) {  // reset all interfaces, cores, and other circuitry on the card

    printf("call to DSResetCard failed \n");
}
```

See Also

DSResetCores, DSResetDevices

DSRunCard, DSRunCores, DSRunDevices

DSInitCard

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore

DSResetCores

UINT DSResetCores (hCard, qwCoreList)

```
HCARD  hCard       // multicore CPU or DSP/analog hardware handle
QWORD  qwCoreList  // value specifying which cores should be reset
```

The DSResetCores function places one or more cores on the multicore CPU or DSP/analog hardware specified by hCard into the reset state. The qwCoreList parameter is used to indicate which cores should be reset.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
qwCoreList	Value containing core list; each bit represents a core, starting with bit 0 for core 0; a 1 causes the core to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the cores specified by qwCoreList were reset successfully. Otherwise, the return value is NULL, indicating the DSResetCores call failed.

Comments

The DSResetCores function will return NULL if qwCoreList is zero, or if qwCoreList specifies cores outside of the actual physical configuration of the multicore CPU or DSP/analog hardware. For example, if qwCoreList specifies more total cores than the card contains, or qwCoreList specifies core numbers outside the range of cores on the card, then DSResetCores will return NULL.

Upon successful return, cores on the multicore CPU or DSP/analog hardware specified by qwCoreList will be in the reset state.

Cores specified by qwCoreList are reset simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if all cores need to be reset simultaneously, it is simpler to use the DSResetCard function.

For some types of hardware, the DSResetCores call may actually return with multicore CPU or DSP devices in the run state, in which case a resident "host interface" code segment will be active on the device. This is necessary in cases where the design of the hardware and/or device does not allow oncard memory to be accessed while the device(s) are held in reset.

For a detailed description of qwCoreList behavior, see section 2.3.2 below, API Core List Parameters.

Note that in all cases, if the DSResetCores function is successful, further operations such as DSReadMem and DSWriteMem can be executed at any time. *If the DSResetCores function is not successful, then further operations will produce undefined results.*

NOTE: DSResetCores should not be called unless a successful DSInitCard function has been completed previously.

C/C++ Example

```
unsigned int uNumDevices = DSGetCardInfo(hCard, DS_GCI_NUMDEVICES);
unsigned int uNumCoresPerDevice = DSGetCardInfo(hCard, DS_GCI_NUMCORESPERDEVICE);

QWORD qwCoreList = pow(2, uNumDevices * uNumCoresPerDevice) – 1;  // set core list to max number of cores on the card
```

```
if (!DSResetCores(hCard, qwCoreList)) {  // reset all cores

    printf("call to DSResetCores failed \n");
}
```

See Also

DSResetCard, DSResetDevices

DSRunCores, DSRunCard, DSRunDevices

DSInitCard

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore software

DSResetDevices

UINT DSResetDevices (hCard, uAttributes, dwDeviceList)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uAttributes	// attributes specifying type of reset
DWORD dwDeviceList	// value specifying which devices should be reset

The DSResetCores function places one or more cores on the multicore CPU or DSP/analog hardware specified by hCard into the reset state. The qwCoreList parameter is used to indicate which cores should be reset.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uAttributes	Attributes specifying type of reset to perform, as listed below:

Value	Meaning
DS_RD_HARDRESET	Perform a hard reset of the device(s), including all interface and peripheral circuitry on the device(s)
DS_RD_SOFTRESET	Perform a soft reset of the device(s). Usually this means only a CPU reset, although behavior can vary widely between device types

dwDeviceList	Value containing device list; each bit represents a device, starting with bit 0 for core 0; a 1 causes the device to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the cores specified by dwDeviceList were reset successfully. Otherwise, the return value is NULL, indicating the DSResetDevices call failed.

Comments

The DSResetDevices function will return NULL if dwDeviceList is zero, or if dwDeviceList specifies devices outside of the actual physical configuration of the multicore CPU or DSP/analog hardware. For example, if dwDeviceList specifies more total devices than the card contains, or dwDeviceList specifies core numbers outside the range of devices on the card, then DSResetDevices will return NULL.

Upon successful return, cores on the multicore CPU or DSP/analog hardware specified by dwDeviceList will be in the reset state.

Devices specified by dwDeviceList are reset simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-access mode. For multicore hardware, if all devices need to be reset simultaneously, and a hard reset is needed, it may be simpler to use the DSResetCard function.

For some types of hardware, the DSResetDevices call may actually return with multicore CPU or DSP devices in a “pseudo run state”, in which case a resident "host interface" code segment will be active on the device. This is necessary in cases where the design of the hardware and/or device does not allow oncard memory to be accessed while the device(s) are held in reset.

Note that in all cases, if the DSResetDevices function is successful, further operations such as DSReadMem and DSWriteMem can be executed at any time. *If the DSResetDevices function is not successful, then further operations will produce undefined results*.

NOTE: DSResetDevices should not be called unless a successful DSInitCard function has been completed previously (typically done once at program start).

C/C++ Example

```
unsigned int uNumDevices = DSGetCardInfo(hCard, DS_GCI_NUMDEVICES);

DWORD dwDeviceList = pow(2, uNumDevices) – 1;  // set device list to max number of devices on the card
dwDeviceList &= 0x5555;

if (!DSResetDevices(hCard, dwDeviceList)) {  // reset all even-numbered devices on the card, starting with device 0

    printf("call to DSResetDevices failed \n");
}
```

See Also

DSResetCard, DSResetCores

DSRunCores, DSRunCard, DSRunDevices

DSInitCard

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore software

 Copyright © Signalogic 1994-2013

DSRunCard

UINT DSRunCard (hCard)

```
HCARD hCard    // multicore CPU or DSP/analog hardware handle
```

The DSRunCard function places all cores on the multicore CPU or DSP/analog hardware specified by hCard into the run state.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call

Returns

The return value is non-zero if all cores were started successfully. Otherwise, the return value is NULL, indicating the DSRunCard call failed.

Comments

Upon successful return, all cores on the multicore CPU or DSP/analog hardware will be in the run state.

All cores are started simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if only certain cores need to be started, use the DSRunCores function.

Note that in all cases, if the DSRunCard function is successful, data transfer operations such as DSReadMem and DSWriteMem may still be executed at any time. *If the DSRunCard function is not successful, then further operations will produce undefined results.*

C/C++ Example

See Also

DSRunCores, DSRunDevices

DSResetCard, DSResetCores, DSResetDevices

DSInitCard

DSDisableCard

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSRunCores

UINT DSRunCores (hCard, qwCoreList)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
QWORD  qwCoreList // value specifying which cores should be run
```

The DSRunCores function places one or more cores on the multicore CPU or DSP/analog hardware specified by hCard into the run state. The qwCoreList parameter is used to indicate which cores should be started.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
qwCoreList	Value containing core list; each bit represents a core, starting with bit 0 for core 0; a 1 causes the core to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the cores specified by qwCoreList were started successfully. Otherwise, the return value is NULL, indicating the DSRunCores call failed.

Comments

The DSRunCores function will return NULL if qwCoreList is zero, or if qwCoreList specifies cores outside of the actual physical configuration of the multicore CPU or DSP/analog hardware. For example, if qwCoreList specifies more total cores than the card contains, or qwCoreList specifies core numbers outside the range of cores on the card, then DSRunCores will return NULL.

Upon successful return, devices on the multicore CPU or DSP/analog hardware specified by qwCoreList will be in the run state.

DSP devices specified by qwCoreList are started simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if all cores need to be started simultaneously, it is simpler to use the DSRunCard function.

For a detailed description of qwCoreList behavior, see section 2.3.2 below, API Core List Parameters.

Note that in all cases, if the DSRunCores function is successful, data transfer operations such as DSReadMem and DSWriteMem can be executed at any time. *If the DSRunCores function is not successful, then further operations will produce undefined results.*

C/C++ Example

See Also

DSRunCard, DSRunDevices

DSResetCores, DSResetCard, DSResetDevices

DSInitCard

DSDisableCard

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSRunDevices

UINT DSRunDevices (hCard,
dwDeviceList)

HCARD hCard // multicore CPU or DSP/analog hardware handle
DWORD dwDeviceList // value specifying which devices should be run

The DSRunDevices function places one or more devices on the multicore CPU or DSP/analog hardware specified by hCard into the run state. The dwDeviceList parameter is used to indicate which devices should be started.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
dwDeviceList	Value containing device list; each bit represents a device, starting with bit 0 for core 0; a 1 causes the device to be included, a 0 causes it to be unaffected

Returns

The return value is non-zero if the devices specified by dwDeviceList were started successfully. Otherwise, the return value is NULL, indicating the DSRunDevices call failed.

Comments

The DSRunDevices function will return NULL if dwDeviceList is zero, or if dwDeviceList specifies devices outside of the actual physical configuration of the multicore CPU or DSP/analog hardware. For example, if dwDeviceList specifies more total devices than the card contains, or dwDeviceList specifies device numbers outside the range of devices on the card, then DSRunDevices will return NULL.

Upon successful return, devices on the multicore CPU or DSP/analog hardware specified by dwDeviceList will be in the run state.

Devices specified by dwDeviceList are started simultaneously, if the multicore CPU or DSP/analog hardware supports a simultaneous-write mode. For multicore hardware, if all devices need to be started simultaneously, it is simpler to use the DSRunCard function.

Note that in all cases, if the DSRunDevices function is successful, data transfer operations such as DSReadMem and DSWriteMem can be executed at any time. *If the DSRunDevices call is not successful, then further operations will produce undefined results.*

C/C++ Example

See Also

DSRunCard, DSRunCores

DSResetCard, DSResetCores, DSResetDevices

DSInitCard

DSDisableCard

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSSetCardBaseAddr

UINT DSSetCardBaseAddr (hCard,
uAddrType,
wBaseAddr)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
UINT  uAddrType   // type of base address to set
WORD  wBaseAddr   // base address value
```

The DSGetCardBaseAddr function sets the current I/O or memory base address for the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uAddrType	Type of base address to set; can be one of the values defined below:

Value	Meaning
DS_GCBA_IOADDR	retrieve I/O base address
DS_GCBA_MEMADDR	retrieve memory base address

wBaseAddr	Value to which to set base address

Returns

The return value is TRUE if the I/O or memory base address was set successfully for the multicore CPU or DSP/analog hardware specified by hCard. Otherwise, the return value is NULL, indicating the DSSetCardBaseAddr call failed.

Comments

DSSetCardBaseAddr can be used to set base addresses for multicore CPU or DSP/analog hardware at any time. Note these values were initialized in the original DSAssignCard call which created hCard. Note also that if either the uIOBaseAddr or uMemBaseAddr parameters in the original DSAssignCard calls were NULL, then as a result of the DSSetCardBaseAddr call, these values are initialized to the current I/O Base Address and Memory Base Address values in the Hardware Setup File.

The DSGetCardBaseAddr call can be used to retrieve multicore CPU or DSP/analog hardware base address values at any time. The values returned are either initialized in the original DSAssignCard call or set subsequently using a DSSetCardBaseAddr call.

C/C++ Example

See Also

DSGetCardBaseAddr

DSAssignCard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Memory Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSSetCardBusType

UINT DSSetCardBusType (hCard, uBusType)

```
HCARD  hCard      // multicore CPU or DSP/analog hardware handle
UINT   uBusType   // type of base address to set
```

The DSSetCardBusType function sets the current bus type for the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uBusType	Bus type to set; can be one of the values defined below:

Value	Meaning
DS_AC_PCXT	PC-XT bus type (deprecated)
DS_AC_PCAT	PC-AT bus type (deprecated)
DS_AC_ENABLE80X86INST	Enable 80x86 instruction set operations (deprecated)
DS_AC_PCIEX1	PCIe bus type, 1 lane
DS_AC_PCIEX4	PCIe bus type, 4 lanes
DS_AC_PCIEX8	PCIe bus type, 8 lanes
DS_AC_PCIEX16	PCIe bus type, 16 lanes

Returns

The return value is TRUE if the bus type was set successfully for the multicore CPU or DSP/analog hardware specified by hCard. Otherwise, the return value is NULL, indicating the DSSetCardBusType call failed.

Comments

Depending on the speed of the host server, and the type of multicore CPU or DSP/analog hardware, there may be cases where specifying a different bus type can make the difference between correct operation of the hardware, or improve performance. For example, a few older multicore CPU or DSP/analog hardware types do not operate reliably when 80x86 type "string move" instructions are used, in which case the DS_AC_ENABLE80X86INST bus type attribute shown above should be avoided. For complete bus type information for specific hardware, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide.

DSSetCardBusType can be used to set bus types for multicore CPU or DSP/analog hardware at any time. Note this value is initialized in the original DSAssignCard call which created hCard. Note also that if the uBusType parameter in the original DSAssignCard calls was NULL, then this value is initialized to the current Bus Type value in the Hardware Setup File.

The DSGetCardBusType call can be used to retrieve multicore CPU or DSP/analog hardware bus type values at any time. The value returned is either that initialized in the original DSAssignCard call or set subsequently using a DSSetCardBusType call.

C/C++ Example

See Also

DSGetCardBusType

DSAssignCard

Related Users Guide Topics

Bus Type Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSSetCoreList

UINT DSSetCoreList (hCard, qwCoreList)

```
HCARD  hCard       // multicore CPU or DSP/analog hardware handle
QWORD qwCoreList   // list of cores to set as currently selected
```

The DSSetCoreList function sets the currently selected core list.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
qwCoreList	List of cores to set as currently selected

Returns

The return value is TRUE if the function is successful.

Comments

For more information about how the currently selected core list works, see section 2.3 below, About Core Lists.

For a detailed description of qwCoreList behavior, see section 2.3.2 below, API Core List Parameters.

Note that the DSSetCoreList function does not actually access hardware. It only accesses an internal memory database used to control and monitor core usage.

NOTE: DSSetCoreList should not be called unless a successful DSInitCard function has been completed previously.

C/C++ Example

```
unsigned int uNumDevices = DSGetCardInfo(hCard, DS_GCI_NUMDEVICES);
unsigned int uNumCoresPerDevice = DSGetCardInfo(hCard, DS_GCI_NUMCORESPERDEVICE);

QWORD qwCoreList = pow(2, uNumDevices * uNumCoresPerDevice) – 1;  // set core list to max number of cores on the card

if (!DSSetCoreList(hCard, qwCoreList)) {  // set currently selected cores
```

```
    printf("call to DSSetCoreList failed \n");
}
```

See Also

DSGetCoreList

DSInitCard, DSInitCores

DSLoadFileCard, DSLoadFileCores

Related Users Guide Topics

section 4.3.1, Hardware and Core Control

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore software

DSWaitForBuffer

UINT DSWaitForBuffer (hCard, wBufferNum, dwFlagAddr, uNotifyType)

```
HCARD  hCard        // multicore CPU or DSP/analog hardware handle
WORD  wBufferNum    // buffer number
DWORD  dwFlagAddr   // flag address in multicore CPU or DSP memory
UINT  uNotifyType   // notification type
```

The DSWaitForBuffer function establishes a DSP Engine wait-for-buffer request with the specified multicore CPU or DSP/analog hardware.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
wBufferNum	Number of buffer to wait for
dwFlagAddr	For user-defined buffers only, address of buffer flag in multicore CPU or DSP memory
uNotifyType	Notification type; can be one of the values defined below:

Value	Meaning
DS_WFB_POLLED	wait request is polled mode
DS_WFB_INT	wait request is interrupt-driven

Returns

The return value is TRUE if the wait-for-buffer request was established successfully. Otherwise, the return value is NULL, indicating the DSWaitForBuffer call failed.

Comments

A wait-for-buffer request is established with a DSP Engine by calling DSWaitForBuffer. Below is a short source code example (see also C/C++ Example below):

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hCard, wBufferNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_BUFRDY messages.

wBufferNum values 0 through 15 are reserved for standard DSP Engine high-level function "data nodes", which are pre-defined sources and sinks of analog input and output data. In these cases, dwFlagAddr and uNotifyType are ignored. For example, wBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1. uNotifyType is ignored for reserved wBufferNum values because DSP Engines may use specialized methods of buffer-checking, which may involve hardware register bits or other optimized techniques (not necessarily multicore CPU or DSP card memory). The concept behind reserved data nodes is that more than one calling application may register a window handle to receive the same buffer ready messages.

Values for wBufferNum not equal to 0 through 15 can be used to set up user-defined buffer ready messages. In these cases, dwFlagAddr and uNotifyType must be specified.

If a uNotifyType value of DS_WFB_POLLED is specified, the DSP Engine will periodically check to see if the specified buffer is ready, and cause WM_DSPENGINE_BUFRDY messages to be sent to hwnd as needed. If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable a host server interrupt, and send a WM_DSPENGINE_BUFRDY message when the interrupt occurs.

For more detailed information, see Buffer Ready Messages from DSP Engines, in Signalogic Software Users Guide section 7.4, Registering Callback Messages from DSP Engines.

C/C++ Example

Below is another source code example taken from dscope.cpp, showing partial code for creating a window and designating it to receive buffer ready messages:

```
// create window that can be used for callback messages
hwnd = CreateWindow(wndclass.lpszClassName,
        "Example DirectCore Test Program",
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL | WS_CLIPCHILDREN | WS_VISIBLE,
        rectScopePos.left,
        rectScopePos.top,
        nWidth,
        nHeight,
        NULL,
        NULL, // no menu
        hInstance,
        NULL);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// scope window is born; turn the card loose...
DSRunCores(hCard, 0x01);  // core 0 only

// register window to receive buffer ready messages, tell engine to wait for input data

DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hCard, 0, NULL, DS_WFB_POLLED);

// no choice now but to follow Alice...
while (GetMessage(&msg, NULL, 0, 0)) {

    TranslateMessage (&msg);
    DispatchMessage (&msg);
};
```

See Also

WM_DSPENGINE_BUFRDY

DSCancelWaitBuffer

DSWaitForFlag

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

Related Users Guide Topics

Buffer Ready Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

DSWaitForFlag

UINT DSWaitForFlag (hCard,
wFlagNum,
dwFlagAddr,
uNotifyType)

```
HCARD  hCard        // multicore CPU or DSP/analog hardware handle
WORD  wFlagNum      // flag number
DWORD  dwFlagAddr   // flag address in multicore CPU or DSP memory
UINT  uNotifyType   // notification type
```

The DSWaitForFlag function establishes a DSP Engine wait-for-semaphore request with the specified multicore CPU or DSP/analog hardware.

See below for function parameter descriptions.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
wFlagNum	Number of flag to wait for
dwFlagAddr	For user-defined semaphores only, address of flag in multicore CPU or DSP memory
uNotifyType	Notification type; can be one of the values defined below:

Value	Meaning
DS_WFB_POLLED	wait request is polled mode
DS_WFB_INT	wait request is interrupt-driven

Returns

The return value is TRUE if the wait-for-flag request was established successfully. Otherwise, the return value is NULL, indicating the DSWaitForFlag call failed.

Comments

A wait-for-semaphore request is established with a DSP Engine by calling DSWaitForFlag. Below is a short source code example (see also C/C++ Example below):

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForFlag(hCard, wFlagNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_FLGRDY messages.

wFlagNum values 0 through 15 are reserved for standard DSP Engine high-level function "semaphore nodes", which are pre-defined status indicators for standard types of analog input and output data and acceleration processing. In these cases, dwFlagAddr and uNotifyType are ignored. For example, an wFlagNum value of 0 indicates accelerated file-based time domain data processing. uNotifyType is ignored for reserved wFlagNum values because DSP Engines may use specialized methods of flag-checking, which may involve hardware register bits or other optimized technique (not necessarily multicore CPU or DSP card memory). The concept behind reserved semaphore nodes is that more than one calling application may register a window handle to receive the same semaphore messages.

Values for wFlagNum not equal to 0 through 15 can be used to set up user-defined semaphore messages. In these cases, dwFlagAddr and uNotifyType must be specified.

If a uNotifyType value of DS_WFB_POLLED is specified, the DSP Engine will periodically check to see if the specified semaphore is ready, and cause WM_DSPENGINE_FLGRDY messages to be sent to hwnd as needed. If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available card-to-host server interrupt, and send a WM_DSPENGINE_FLGRDY message to hwnd when the interrupt occurs.

For more detailed information, see Semaphore Messages from DSP Engines, in Signalogic Software Users Guide section 7.4, Registering Callback Messages from DSP Engines.

C/C++ Example

See Also

WM_DSPENGINE_FLGRDY

DSCancelWaitFlag

DSWaitForBuffer

DSReadMem, DSWriteMem, DSReadMemEx, DSWriteMemEx

Related Users Guide Topics

Semaphore Messages from DSP Engines, in section 7.4 Registering Callback Messages from DSP Engines

DSWriteCardReg

WORD DSWriteCardReg (hCard,
wRegOffset,
wVal)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
WORD wRegOffset	// offset from register base address
WORD wVal	// value to write

The DSWriteCardReg function writes a value to the specified register on the multicore CPU or DSP/analog hardware indicated by hCard.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
wRegOffset	Offset from I/O base address of multicore CPU or DSP/analog hardware
wVal	Value to write to multicore CPU or DSP/analog hardware register

Returns

The return value is TRUE if the value was written to the multicore CPU or DSP/analog hardware register successfully. Otherwise, the value is NULL, indicating the DSWriteCardReg function failed, and the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines should be checked (see C/C++ source code example below).

Comments

In cases where directly accessing multicore CPU or DSP/analog hardware is unavoidable, the DSReadCardReg and DSWriteCardReg functions may prove convenient.

C/C++ Example

```
if (!DSWriteCardReg(hCard, wRegOffset, 0x59a2)) {

    HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE);
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();

    if (nErrorType != NULL) {

        // do error processing...
    }
}
```

See Also

DSReadCardReg

DSSetCardBaseAddr

DSAssignCard

Related Users Guide Topics

I/O Base Address Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Related Documentation Sections

Signalogic Software Users Guide section 5.0.2, DSP Source Code Interfaces for Hypersignal

Signalogic Software Users Guide section 5.0.3, DSP Source Code Interfaces for Hypersignal in Engine Mode

Hypersignal-Macro Series Hardware Reference Guide chapter 4, DSP Source Code Interface

Hypersignal-Macro Series Hardware Reference Guide Appendix A, WriteDSPVal Constants

See Also

DSGetProperty

DSSetProperty

DSWriteMem

UINT DSWriteMem (hCard,
uMemType,
dwDeviceMemAddr,
uBitWidth,
pMem,
dwCount)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uMemType	// multicore CPU or DSP memory type
DWORD dwDeviceMemAddr	// multicore CPU or DSP memory start address
UINT uBitWidth	// data element width in multicore CPU or DSP memory
PMEM pMem	// pointer to buffer to receive data
DWORD dwCount	// number of data elements to transfer

The DSWriteMem function transfers one or more data elements from host server memory defined by the calling application to multicore CPU or DSP/analog hardware memory.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uMemType	Type of multicore CPU or DSP memory to which to transfer data elements, as defined below:

Value	Meaning
DS_RM_DIRECTACCESS	Currently deprecated; used with some legacy hardware
DS_RM_L2	Core specific L2 memory (typically onchip memory)
DS_RM_LINEAR_PROGRAM	Linear program memory
DS_RM_LINEAR_DATA	Linear data memory
DS_RM_LINEAR_DATA_RT	Linear data memory fast transfer while device is in run state (used only for some legacy DSPs)
DS_RM_MCSM	Multicore shared memory (typically onchip memory)
DS_EXTSDRAM	Device external SDRAM memory, for example DDR3 SDRAM
DS_RM_VECTOR_DATA_X	Vector X: data memory
DS_RM_VECTOR_DATA_Y	Vector Y: data memory

dwDeviceMemAddr	Starting address in multicore CPU or DSP data memory for transfer
uBitWidth	Data element width in multicore CPU or DSP memory for transfers, as defined below:

Value	Meaning
DS_RM_SIZE8	Device memory data element width is 1 byte

DS_RM_SIZE16	Device memory data element width is WORD
DS_RM_SIZE24	Device memory data element width is 3 bytes
DS_RM_SIZE32	Device memory data element width is DWORD or float

pMem	Pointer to calling application memory which contains data elements to transfer
dwCount	Number of data elements to transfer

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSWriteMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

The DSReadMem and DSWriteMem functions are flexible, general-purpose functions intended to transfer data between host server memory and oncard or onchip multicore CPU or DSP/analog hardware memory. Because of the wide variety of multicore CPU or DSP/analog hardware memory types, and the flexibility of the DSReadMem and DSWriteMem APIs, there are several items to keep in mind:

1) For some types of multicore CPU or DSP/analog hardware, uBitWidth values of DS_RM_SIZE8 and DS_RM_SIZE24 are not supported. In cases where a uBitWidth value is not supported, the transfer defaults to the next higher data element size.

2) If the multicore CPU or DSP hardware supports multicore shared memory, L2 core specific memory, or external SDRAM memory, then the DS_RM_MCSM, DS_RM_L2, and DS_RM_EXTSDRAM memory type attributes can be used, and the DS_RM_LINEAR_xxx attributes can be ignored. Note that in some cases the physical device address may make an attribute superfluous; for instance, for C66x hardware if DS_RM_EXTSDRAM is combined with an external device address higher than 0x80000000, then the attribute is redundant.

3) For 32-bit multicore CPU and DSP devices which support byte- or word-addressable memory, addressing used by DSReadMem and DSWriteMem appears as if device memory is accessible as contiguous locations each of uBitWidth bits. For example, if a device supports byte-addressable memory, then using a uBitWidth value of DS_RM_SIZE16 would cause access to occur at the *byte address* of dwDeviceDataAddr∗2, and a uBitWidth value of DS_RM_SIZE32 would cause access to occur at the byte address of dwDeviceMemAddr∗4 (or the *word address* of dwDeviceMemAddr∗2). Examples of byte-addressable devices include the Texas Instruments C64x and C66x series, AT&T DSP32C and DSP3210, and Texas Instruments TMS320C32.

4) Currently, no dynamic data conversion between integer and floating-point or IEEE floating-point and native device floating-point formats is made by DSReadMem or DSWriteMem. For example, if a calling application is transferring 64-bit floating-point data, the multicore CPU or DSP program may need to “know” whether to convert this data to its native 32-bit floating-point format prior to processing.

5) Currently, the DS_RM_VECTOR_DATA_X and DS_RM_VECTOR_DATA_Y uMemType attributes are only supported for Freescale DSP5600x and DSP9600x devices. DS_RM_VECTOR_DATA_X and DS_RM_LINEAR_DATA are equivalent.

C/C++ Example

The source code example below transfers quantized sine wave (1000 kHz sine sampled at 100 kHz) data to multicore CPU or DSP data memory address 0x800 from a calling application data buffer created by the Windows GlobalAlloc function. Note that the actual type of memory receiving the data could be anything from 16-bit, 24-bit, or 32-bit data—the host software buffer contains only quantized 16-bit values stored in 32-bit signed integers.

```
HGLOBAL hBuf;
long* pBuf;
WORD wBuflen;
DWORD dwDeviceMemAddr;

  wDeviceMemAddr = 0x800;
```

```
hBuf = GlobalAlloc(GMEM_MOVEABLE, wBuflen*sizeof(long));
pBuf = (long*)GlobalLock(hBuf);
for n = 0; n<wBufLen-1; n++) pBuf[n] = (long)32767*sin(2*pi*n*1000/10000);

UINT uStatus = DSWriteMem(hCard, DS_RM_LINEAR_DATA, dwDeviceMemAddr, DSGetWordLength(hCard), pBuf, wBuflen);

if (!uStatus) {

    HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSWriteMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that the DSGetWordLength function is used to transfer data using a data element width equal to the natural word length of the multicore CPU or DSP device. This method is reliable mainly for integer data, and not floating-point data, because of the varying storage representations of floating-point (or fractional integer format) data used by popular multicore CPU and DSP devices.

See Also

DSReadMem, DSReadMemEx, DSWriteMemEx

DSGetProperty

DSSetProperty

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE, DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

 Copyright © Signalogic 1994-2013

DSWriteMemEx

UINT DSWriteMemEx (hCard,
uMemType,
dwDeviceMemAddr,
uBitWidth,
pMem,
dwCount,
qwCoreList)

HCARD hCard	// multicore CPU or DSP/analog hardware handle
UINT uMemType	// multicore CPU or DSP memory type
DWORD dwDeviceMemAddr	// multicore CPU or DSP memory start address
UINT uBitWidth	// data element width in multicore CPU or DSP memory
PMEM pMem	// pointer to buffer to receive data
DWORD dwCount	// number of data elements to transfer
QWORD qwCoreList	// list of cores from which to read

The DSWriteMemEx function transfers one or more data elements from host server memory defined by the calling application to multicore CPU or DSP/analog hardware memory.

Parameter	Description
hCard	Multicore CPU or DSP/analog hardware handle previously obtained from DSAssignCard call
uMemType	Type of multicore CPU or DSP memory to which to transfer data elements, as defined below:

Value	Meaning
DS_RM_DIRECTACCESS	Currently deprecated; used with some legacy hardware
DS_RM_L2	Core specific L2 memory (typically onchip memory)
DS_RM_LINEAR_PROGRAM	Linear program memory
DS_RM_LINEAR_DATA	Linear data memory
DS_RM_LINEAR_DATA_RT	Linear data memory fast transfer while device is in run state (used only for some legacy DSPs)
DS_RM_MCSM	Multicore shared memory (typically onchip memory)
DS_RM_EXTSDRAM	Device external SDRAM memory, for example DDR3 SDRAM
DS_RM_VECTOR_DATA_X	Vector X: data memory
DS_RM_VECTOR_DATA_Y	Vector Y: data memory

dwDeviceMemAddr	Starting address in multicore CPU or DSP data memory for transfer
uBitWidth	Data element width in multicore CPU or DSP memory for transfers, as defined below:

Value	Meaning
DS_RM_SIZE8	Device memory data element width is 1 byte
DS_RM_SIZE16	Device memory data element width is WORD
DS_RM_SIZE24	Device memory data element width is 3 bytes
DS_RM_SIZE32	Device memory data element width is DWORD or float

pMem	Pointer to calling application memory which contains data elements to transfer
dwCount	Number of data elements to transfer
qwCoreList	List of cores for which memory should be written

Returns

The return value is a TRUE if the transfer was completed successfully. Otherwise, the return value is NULL, indicating the DSWriteMem call failed. For further error checking, use the DSGetEngineErrorStatus and DSGetHWLibErrorStatus routines (see C/C++ source code example below).

Comments

See the comments in the DSReadMem and DSWriteMem function descriptions.

For a detailed description of qwCoreList behavior, see section 2.3.2 below, API Core List Parameters.

DirectCore / driver software attempts to perform a “broadcast mode” write as allowed by the hardware design, minimizing the number of actual physical write operations.

If multiple cores are specified and the device memory address does not specify core-specific memory -- for example specifying onchip shared memory or external device memory shared by all cores – then results are undefined, although DirectCore / driver software may attempt to recognize the redundancy and physically write data only once (this behavior is not guaranteed on all systems).

C/C++ Example

The source code example below transfers quantized sine wave (1000 kHz sine sampled at 100 kHz) data to multicore CPU or DSP data memory address 0x800 from a calling application data buffer created by an operating-system dependent memory allocation function. In the example same data is transferred to core-specific memory for each core specified by the qwCoreList parameter. Note that the actual type of memory receiving the data could be anything from 16-bit, 24-bit, or 32-bit data--the host software buffer contains quantized 16-bit values stored as 32-bit signed integers.

```
HGLOBAL hBuf;
long* pBuf;
WORD wBuflen;
DWORD dwDeviceMemAddr = 0x800;
UINT uNumCores = 8;
QWORD qwCoreList = pow(2, uNumCores) – 1;
UINT uStatus;

hBuf = GlobalAlloc(GMEM_MOVEABLE, wBuflen*sizeof(long));
pBuf = (long*)GlobalLock(hBuf);
for n = 0; n<wBufLen-1; n++) pBuf[n] = (long)32767∗sin(2∗pi∗n∗1000/10000);

uStatus = DSWriteMemEx(hCard, DS_RM_LINEAR_DATA, dwDeviceMemAddr, DSGetWordLength(hCard), pBuf, wBuflen, qwCoreList);

if (!uStatus) {

    HENGINE hEngine = DSGetCardInfo(hCard, DS_GCI_ENGINEHANDLE));
    int nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
```

```
    itoa(nErrorType, tmpstr, 10);
    MessageBox(NULL, tmpstr, "DSWriteMem error occurred", MB_OK);
}
GlobalUnlock(hBuf);  // release mem
```

Note in the above example that the DSGetWordLength function is used to transfer data using a data element width equal to the natural word length of the multicore CPU or DSP device. This method is reliable mainly for integer data, and not floating-point data, because of the varying storage representations of floating-point (or fractional integer format) data used by popular multicore CPU and DSP devices.

C/C++ Example 2

```
UINT uNumCores = 48;
QWORD qwCoreList = pow(2, uNumCores) – 1;
DWORD dwDeviceMemAddr = 0x800;
DWORD dwVal = 0x12345678;
UINT uStatus;

uStatus = DSWriteMemEx(hCard, DS_RM_LINEAR_DATA, dwDeviceMemAddr, DSGetWordLength(hCard), dwVal, 1, qwCoreList);

if (!uStatus) {

    // do error processing…
```

See Also

DSReadMem, DSReadMemEx, DSWriteMem

DSGetProperty, DSSetProperty

DSWaitForBuffer

WM_DSPENGINE_BUFRDY

DSGetWordLength

DSDSPToIEEE, DSIEEEToDSP

Related Users Guide Topics

section 4.3.4, Multicore CPU and DSP Memory Data Transfers

Example C/C++ Code: Processing Buffer Ready Messages, in section 4.2.4, Example Code: DirectCore Software

2.2.2 File Library Functions

File library provides file management support for popular audio and video waveform files., including file header view and management, format conversion, and file transfer to/from multicore CPU or DSP hardware.

2.2.3 Engine Manager Functions

Below are listed functions in the Engine Manager API. For information on how to include Engine Manager (enmgr.dll or enmgr32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic Software Users Guide section 7.1, Engine Manager Access.

Engine Manager Functions

DSAddEngMgrStatusLine

DSEngineOpen

DSEngineClose

DSExecEngineFunction
DSGetEngineErrorStatus
DSGetEnginePath
DSGetEngineUsage
DSGetEngineState
DSGetEngMgrErrorStatus
DSHideEngMgrStatusWindow
DSRegisterEngineMsgWnd
DSSendEngineCommand
DSSetEngineTemplatePath
DSSetEngineWaveformPath
DSShowEngMgrStatusWindow
DSUnregisterEngineMsgWnd

DSAddEngMgrStatusLine

HWND DSAddEngMgrStatusLine (pszText)

LPCSTR pszText // pointer to text string to add to error/status window

The DSAddEngMgrStatusLine function adds a line of text to the Engine Manager error/status listbox. Text is always added to the end of any text currently in the listbox, regardless of whether the error/status listbox is currently enabled or not.

Parameter	Description
pszText	Pointer to null-terminated text string to add to Engine Manager error/status listbox

Returns

The return value is the handle of the error/status listbox window, if successful. Otherwise, the return value is NULL, indicating that the error/status box could not be created or updated with new information.

Comments

The error/status listbox is created if no text has previously been added or displayed.

Text is added to the Engine Manager error/status listbox regardless of whether the listbox is visible or not. Text is always added to the end of the current listbox contents, which are scrolled up, if necessary, to ensure the most recently added text is in view.

C/C++ Example

```
hwnd = DSAddEngMgrStatusLine("processing buffer-ready message");
if (hwnd != NULL) {

    SetMenu(hwnd, hMenu);  // add application-specific menu to status box
}
```

See Also

DSHideEngMgrStatusWindow

DSShowEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSEngineOpen

HENGINE DSEngineOpen (pszEnginePath, pszWaveformPath)

```
LPCSTR  pszEnginePath      // path where engine is located
LPCSTR  pszWaveformPath    // path of engine waveform files
```

The DSEngineOpen function opens and initializes the specified DSP Engine, and returns an engine handle which can be used in numerous other Engine Manager functions, as well as some Hardware Library and Signal Processing Library functions. Some pre-defined constants can be specified in place of **szEnginePath**, as shown below:

Parameter	Description
pszEnginePath	pointer to null-terminated string containing partially- or fully-qualified path of DSP Engine to open; alternatively, the constants defined below can be used:

Value	Meaning
DS_EO_HSM	search for and open Hypersignal-Macro as DSP Engine
DS_EO_HSA	search for and open Hypersignal-Acoustic as DSP Engine
DS_EO_HSMNRT	search for and open Hypersignal-Macro NRT as DSP Engine
DS_EO_MAT	search for and open MATLAB as DSP Engine

pszWaveformPath	pointer to null-terminated string containing fully-qualified path location of waveform files which might be accessed by the DSP Engine

Returns

If successful, the return value is non-zero, representing a valid engine handle. Otherwise, the return value is NULL, indicating that the engine could not be located (path was not valid), or the engine could not be initialized. For further error checking, use the DSGetEngMgrErrorStatus routine (see C/C++ source code example below).

Comments

A partially specified engine path will cause the DSEngineOpen function to assume an engine type and perform a rudimentary search. For example, specifying only "matlab" in the engine path will cause the DSEngineOpen function to attempt to open :x:\matlab\bin\matlab.exe", where "x" is the drive location of the Engine Manager library.

The waveform path is not checked for valid waveform files, other than to ensure that the path actually exists.

The DSEngineOpen call opens DSP Engines such as Hypersignal-Macro series or MATLAB programs and places them in command mode, ready to accept instructions. DSP Engines are opened so as not to be visible (and conserve memory by not

loading graphics drivers or other user-interface objects, if possible). Internally, DSP Engines are opened and initialized using either a Windows WinExec call or functions inside shell.dll. If the WinExec call is used, the "show state" parameter is set to SW_HIDE, and in the case of DOS engines, a .pif file may be associated with the engine.

For more information, see also Signalogic Software Users Guide section 7.2.1, Opening DSP Engines.

C/C++ Example

```
HENGINE hEngine;

hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

See Also

DSEngineClose

DSGetEngineUsage

DSSetEngineWaveformPath

DSSetEngineTemplatePath

DSExecEngineFunction

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

section 7.2.2 Sending DSP Engine Commands

section 7.3 Executing Functions in DSP Engines

DSEngineClose

UINT DSEngineClose (hEngine)

HENGINE hEngine // engine handle

The DSEngineClose function closes the specified DSP Engine and invalidates the engine handle.

Parameter	Description
hEngine	Handle to engine obtained from previous DSEngineOpen call

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the engine could not be closed. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

The Engine Manager library is designed to close any engines still open when it is unloaded from the system. Unloading occurs only when all applications which implicitly or explicitly loaded Engine Manager have terminated. However, for each application it is always a good idea to call DSEngineClose for any currently active engine handles just prior to terminating.

This avoids unnecessary consumption of system resources should Engine Manager not be unloaded immediately because other applications are still using it.

Also it is good programming practice to set the hEngine parameter to NULL after a successful DSEngineClose, to avoid the possibility of accidentally using an engine handle value in a Hardware Library call which might have been reassigned to another engine, in use by another application.

For more information details about obtaining engine handles using the DSEngineOpen call, see also Signalogic Software Users Guide section 7.2.1, Opening DSP Engines.

C/C++ Example

```
if (!DSEngineClose(hEngine)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
else hEngine = NULL;
```

See Also

DSEngineOpen

DSGetEngineUsage

DSSetEngineWaveformPath

DSSetEngineTemplatePath

DSExecEngineFunction

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

section 7.2.2 Sending DSP Engine Commands

section 7.3 Executing Functions in DSP Engines

DSExecEngineFunction

UINT DSExecEngineFunction (hEngine, pszCommandStr, uMode)

HENGINE hEngine	// engine handle
LPCSTR pszCommandStr	// command string of function to execute
UINT uMode	// mode of execution

The DSExecEngineFunction call sends a text string containing a function, along with any associated input parameters, to the specified engine for execution. The command string must be null-terminated and can be as long as needed.

Parameter	Description
hEngine	Engine handle which was previously obtained via call to DSEngineOpen

pszCommandStr Pointer to null-terminated command string containing engine function to be executed, and any associated input parameters

uMode Mode in which the engine function should be executed; one of the possible pre-defined values listed below should be specified:

Value	Meaning
DS_EEF_ASYNC	execute engine function asynchronously with respect to calling application (calling application continues to execute simultaneously); this is the default used if uMode is left NULL
DS_EEF_SYNC	execute engine function synchronously with respect to calling application (calling application waits for engine function to complete)

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the command could not be executed. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

DSExecEngineFunction is a general purpose function which allows DSP Engine functions to be invoked from user-defined calling application programs. This facility is not to be confused with DSP Engine functions, or combinations of functions, which are executed automatically and transparently as a result of standard API calls inside the DirectCore libraries.

The purpose of the DSExecEngineFunction call is to allow engine functions to be called directly, assuming knowledge of the syntax and parameters of the engine call.

Either synchronous or asynchronous operation can be specified using the uMode parameter. Synchronous operation means the calling application waits for the engine function to complete; asynchronous operation means the calling application continues to execute simultaneously.

C/C++ Example 1

Below is an example which evaluates a difference equation using Hypersignal-Macro software as the engine. The difference equation creates the output waveform file "sp_avg" from the input waveform file "speech":

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_HSM, "c:\\hsmacro\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "f_diffeq(\"sp_avg\", \"y[n] = y[n] + (x1[n] – x1[n-M])/Fr\", \"speech\", \"\", 0, 0, 0)", DS_EEF_ASYNC);
```

In the above example, the difference equation was executed asynchronously, meaning that the calling application continued to run at the same time. In such a case, the calling application can register a window using the DSRegisterEngineMsgWnd function to receive a "program done" message when the function is complete.

Note the use of "\"" to specify quote marks within the command string for string parameters required by the f_diffeq engine function. For a list of DirectCore software "f_xxxx" engine functions, see section 2.1 above, DSP Engine Functions.

C/C++ Example 2

The example below uses a combination of an engine function and a built-in MATLAB function to execute a simple FFT:

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_MAT, "c:\\matlab\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "x = f_load(\"speech.wav\")", DS_EEF_SYNC);
DSExecEngineFunction(hEngine, "y = fft(x)", DS_EEF_ASYNC);
```

The above example shows the DirectCore software f_load function being used to load the time domain waveform file "speech.wav". Note also this is done synchronously, meaning the calling application would wait for the engine to complete the f_load function. The second DSExecEngineFunction simply passes a command string invoking a MATLAB Signal Processing Toolbox function directly, allowing the calling application to continue executing by specifying the DS_EEF_ASYNC constant. In such a case, the calling application can register a window using the DSRegisterEngineMsgWnd function to receive a "program done" message when the engine function is complete.

C/C++ Example 3

The example below shows the f_fft function being used to accomplish the same result as shown in Example 2 above, but in one step:

```
HENGINE hEngine;
int nErrorType;

hEngine = DSEngineOpen(DS_EO_MAT, "c:\\matlab\\waveform");

if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}

DSExecEngineFunction(hEngine, "y = f_fft(\"speech.wav\")", DS_EEF_ASYNC);
```

See Also

DSRegisterEngineMsgWnd

DSSendEngineCommand

DSSetEngineTemplatePath

DSAcquireWvfrmFile

DSGenerateWvfrmFile

WM_DSPENGINE_PROGSTATUS

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

section 7.4 Registering Callback Messages from DSP Engines

Program Status Messages from DSP Engines (inside section 7.4)

section 7.2.2 Sending DSP Engine Commands

DSGetEngineErrorStatus

UINT DSGetEngineErrorStatus (hEngine)

HENGINE hEngine // engine handle

The DSGetEngineErrorStatus function retrieves the current error status for the specified engine.

Parameter	Description
hEngine	Handle to engine obtained from previous DSEngineOpen call

Returns

The return value indicates the current error condition for the specified engine, if one exists; otherwise, the return value is NULL. For DirectCore functions which require an engine handle and which return NULL, DSGetEngineErrorStatus should be checked first, followed by DSGetEngMgrErrorStatus (see C/C++ source code example below).

Comments

The Engine Manager (enmgr.dll or enmgr32.dll) error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Engine Manager fail, there are two levels of error possibilities: the library itself (or "base level"), and engine-related. Engine-related error conditions are returned by the DSGetEngineErrorStatus call, while base-level error conditions are returned by DSGetEngMgrErrorStatus.

A current error condition is maintained for each active DSP Engine. If an error condition occurs but is not retrieved, and further engine-related calls are made which cause *another* error condition, DSGetEngineErrorStatus returns only the most recent error condition.

Engine Manager also maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from DirectCore library calls.

For more information, see Signalogic Software Users Guide sections 7.5.1, Internal Error Handling in Engine Manager, 7.5.2, Engine Error Handling, and 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSAssignCard call, which is located inside Hardware Library:

```
int nErrorType;

if (!DSAssignCard(hEngine, szCardDesignator, uBusType, uIOBaseAddr, uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the DSGetEngineErrorStatus function should only be used with a valid engine handle. Otherwise, it is possible to incur an "invalid engine handle" error, which may overwrite a previously existing error condition. See **Comments** above.

See Also

DSGetEngMgrErrorStatus

DSGetHWMgrErrorStatus

DSGetHWLibErrorStatus

DSEngineOpen

Related Users Guide Topics

section 7.5 Error Handling in Engine Manager

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSGetEnginePath

UINT DSGetEnginePath (hEngine, pszEnginePath)

```
HENGINE  hEngine        // engine handle
LPSTR  pszEnginePath    // engine path to be filled in and returned
```

The DSGetEnginePath function retrieves the path for the specified DSP Engine and returns the value in pszEnginePath.

Parameter	Description
hEngine	Handle to engine obtained from previous DSEngineOpen call
pszEnginePath	Pointer to string variable which will be filled in with the specified engine path and returned as a null-terminated string

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below).

Comments

The string variable pointed to by pszCardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 144 characters is required (see C/C++ source code example below).

Essentially, the DSGetEnginePath function should return the same path used in the original DSEngineOpen call. The value might be different if a pre-defined constant or partial path was used in the DSEngineOpen call.

C/C++ Example

```
char szEnginePath[144];

if (!DSGetEnginePath(hEngine, szEnginePath)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

See Also

DSEngineOpen

DSSetEngineWaveformPath

DSSetEngineTemplatePath

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngineState

UINT DSGetEngineState (hEngine)

HENGINE hEngine // engine handle

The DSGetEngineState function retrieves the current state of the specified DSP Engine.

Parameter	Description
hEngine	Handle to engine obtained from previous DSEngineOpen call

Returns

The return value is NULL if the engine handle was invalid. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines (see C/C++ source code example below). Otherwise, the return value is as defined below:

Value	Meaning
DS_ES_RUNNING	Engine is in running (busy) state
DS_ES_IDLE	Engine is in idle state
DS_ES_STOPPENDING	Engine is running, but a stop (shutdown) command is pending
DS_ES_STOPPED	Engine is stopped (shutdown)

Comments

The DSGetEngineState call can be used to learn the current state for a specified DSP Engine. Note that state changes can be determined dynamically by using DSRegisterEngineMsgWnd to register a window to receive WM_DSPENGINE_ENGINESTATE messages.

C/C++ Example

```
char szText[50];
short int nErrorType;

switch (DSGetEngineState(hEngine)) {

    case DS_ES_RUNNING:

        _fstrcpy(szText, "Running");
        break;

    case DS_ES_IDLE:

        _fstrcpy(szText, "Idle");
        break;
```

```
    case DS_ES_STOPPENDING:

        _fstrcpy(szText, "Stop Pending");
        break;

    case DS_ES_STOPPED:

        _fstrcpy(szText, "Shutdown");
        break;

    default:

        _fstrcpy(szText, "Error in DSGetEngineState Call");
        nErrorType = DSGetEngineErrorStatus(hEngine);
        if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();

        // do error processing...

        break;
}
MessageBox(NULL, szText, "Current Engine State", MB_OK);
```

See Also

WM_DSPENGINE_ENGINESTATE

DSRegisterEngineMsgWnd

DSEngineOpen

DSEngineClose

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngineUsage

UINT DSGetEngineUsage ( )

The DSGetEngineUsage function returns the number of currently opened DSP Engines by the calling application.

Parameter	Description
None	

Returns

The return value is zero if no engines are currently opened by the calling application. Otherwise, the return value is a positive value indicating the number of engines currently open.

Comments

After invoking a DSEngineClose function for each open engine, DSGetEngineUsage should return 0.

One possible use for this call is in case of catastrophic engine failure, when it may be important to learn if one or more engines has been forced to close due to internal Engine Manager error processing.

C/C++ Example

```
UINT uNumEngines;
char szText[30];

uNumEngines = DSGetEngineUsage();
itoa(uNumEngines, szText, 10);
MessageBox(NULL, szText, "Number of open engines", MB_OK);
```

See Also

DSEngineOpen

DSEngineClose

Related Users Guide Topics

section 7.2 Controlling DSP Engines

section 7.2.1 Opening DSP Engines

DSGetEngMgrErrorStatus

UINT DSGetEngMgrErrorStatus ( )

The DSGetEngMgrErrorStatus function retrieves the current base-level error status for the Engine Manager library(enmgr.dll or enmgr32.dll).

Parameter	Description
None	

Returns

The return value indicates the current Engine Manager library error condition, if one exists; otherwise, the return value is NULL. DSGetEngineErrorStatus should be checked first after DirectCore call which require an engine handle and which return NULL, followed by DSGetEngMgrErrorStatus (see C/C++ source code example below).

Comments

The Engine Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Engine Manager fail, there are two levels of error possibilities: the library itself (or "base level"), and engine-related. Engine-related error conditions are returned by the DSGetEngineErrorStatus call, while base-level error conditions are returned by DSGetEngMgrErrorStatus.

A single, base-level current error condition variable is maintained internally in the Engine Manager library. If an error condition occurs but is not retrieved, and further calls are made which cause *another* base-level library error condition, DSGetEngMgrErrorStatus returns only the most recent error condition.

Engine Manager also maintains an "error/status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from all DirectCore library calls.

For more information, see Signalogic Software Users Guide sections 7.5.1, Internal Error Handling in Engine Manager, 7.5.2, Engine Error Handling, and 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSAssignCard call, which is located inside Hardware Library:

```
int nErrorType;

if (!DSAssignCard(hEngine, szCardDesignator, uBusType, uIOBaseAddr, uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the DSGetEngineErrorStatus function is used first, since a valid engine handle was involved in the function call which returned NULL. Otherwise, for calls such as DSEngineOpen which do not require an engine handle, DSGetEngMgrErrorStatus should be checked first. Note also that it is possible for error conditions to be overwritten if consecutive function calls are made without error-checking. See **Comments** above.

See Also

DSGetEngineErrorStatus

DSGetHWMgrErrorStatus

DSGetHWLibErrorStatus

DSEngineOpen

Related Users Guide Topics

section 7.5 Error Handling in Engine Manager

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSHideEngMgrStatusWindow

void DSHideEngMgrStatusWindow ( )

The DSHideEngMgrStatusWindow function hides the Engine Manager error/status listbox window, if not already hidden. The window is not destroyed, nor is its current contents and text display altered.

Parameter	Description
None	

Returns

No return value.

Comments

At least one call to DSAddEngMgrStatusLine must occur before the error/status listbox is created.

By default, the error/status listbox is not visible. C/C++, Visual Basic, MATLAB, and LabView programs using Engine Manager (part of DirectCore software) must call DSShowEngMgrStatusWindow at some point to make the listbox visible.

C/C++ Example

```
if (MessageBox(NULL, "hide error/status window?", "??", MB_YESNO) == IDYES) {

    DSHideEngMgrStatusWindow();
}
```

See Also

DSAddEngMgrStatusLine

DSShowEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSRegisterEngineMsgWnd

UINT DSRegisterEngineMsgWnd (hEngine, uMsgType, hwnd)

```
HENGINE  hEngine     // engine handle
UINT  uMsgType       // message type(s)
HWND  hwnd           // handle of window to register
```

The DSRegisterEngineMsgWnd function registers a window or control handle to receive one or more types of callback messages. A combination of messages can be specified using the pre-defined constants listed below.

Parameter	Description
hEngine	Engine handle which was previously obtained via call to DSEngineOpen
uMsgType	One or message types for which hwnd should be registered, as defined below:

Value	Meaning
DS_REMW_SETENGINESTATUSMSG	receive engine status messages
DS_REMW_SETPROGSTATUSMSG	receive engine program and function status messages
DS_REMW_SETDSPDATARDYMSG	receive buffer ready and flag ready messages
DS_REMW_SETSTATISTICSMSG	receive statistic and parameter update messages
DS_REMW_SETBUFNUMMSG	receive buffer number update messages
DS_REMW_SETLEVELMSG	receive data level messages
DS_REMW_SETALLMSG	receive all types of callback messages

Hwnd	window or control handle to be registered

 Copyright © Signalogic 1994-2013

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, the window handle was invalid, or one or more message types could not be registered. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

See section 6.1, Library Messages, for information about callback messages which can be received by application programs. The enmgr.h include file supplied with DirectCore and DirectCore software lists all callback message types.

C/C++ Example

```
if (!DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG | DS_REMW_SETDSPDATARDYMSG, hwnd)) {

    MessageBox(NULL, "error in DSRegisterEngineMsgWnd", "", MB_OK);
}
```

See Also

DSUnregisterEngineMsgWnd

DSWaitForBuffer

DSWaitForFlag

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

DSSendEngineCommand

UINT DSSendEngineCommand (hEngine,
wCommand)
uMode)

```
HENGINE  hEngine     // engine handle
WORD  wCommand       // command to send
UINT  uMode          // mode of execution
```

The DSSendEngineCommand function sends commands directly to the specified DSP Engine. Commands can be pre-defined constants defined below, or ASCII or user-defined values as needed.

Parameter	Description
hEngine	Engine handle which was previously obtained via call to DSEngineOpen

wCommand Command to send to engine; possible pre-defined command values are listed below:

Value	Meaning
DS_SEC_ABORT	instruct engine to abort (quit)
DS_SEC_IDLE	instruct engine to enter idle mode
DS_SEC_PAUSE	suspend engine (only valid for DOS engines)
DS_SEC_RESUME	resume engine (only valid for DOS engines)

uMode Mode in which command should be sent; one of the possible pre-defined values listed below should be used:

Value	Meaning
DS_EEF_ASYNC	Send command asynchronously with respect to calling application (calling application continues to execute simultaneously as engine executes command); this is the default used if uMode is left NULL
DS_EEF_SYNC	Send command synchronously with respect to calling application (calling application waits for engine to complete command)

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the command was not sent. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Pre-defined wCommand constants described above are translated by Engine Manager into correct DSP Engine commands. ASCII printable or control characters are sent to the engine without modification. For example the calling application could send command strings to MATLAB by using the DSSendEngineCommand function in a loop which sent each character of the string, and replaced the terminating NULL with a 0x0a value (linefeed).

Pre-defined wCommand constants listed above are encoded as negative values; to ensure that a calling application does not sent commands which will be translated by Engine Manager, consult the enmgr.h include file.

Either synchronous or asynchronous command execution can be specified using the uMode parameter. Synchronous operation means the calling application waits for the engine to execute the command; asynchronous operation means the calling application continues to execute simultaneously.

NOTE: Currently, a uMode value of DS_EEF_ASYNC is only active for the DS_SEC_IDLE and DS_SEC_ABORT commands, which initialize a wait for a WM_DSPENGINE_PROGSTATUS message with DS_PS_PROGDONE parameter.

C/C++ Example 1

```
DSSendEngineCommand(hEngine, DS_SEC_IDLE);  // put engine in idle mode
```

C/C++ Example 2

```
char szCommand[100];
int i;

for (i=0; i<_fstrlen(szCommand); i++) {

    // send command text string

    DSSendEngineCommand(hEngine, szCommand[i], NULL);  // asynchronous
```

```
}
DSSendEngineCommand(hEngine, 0x0a, NULL);  // send Enter key
```

See Also

DSExecEngineFunction

WM_DSPENGINE_PROGSTATUS

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

DSSetEngineTemplatePath

UINT DSSetEngineTemplatePath (hEngine, pszTemplatePath)

HENGINE hEngine	// engine handle
LPCSTR pszTemplatePath	// path for engine template files

The DSSetEngineTemplatePath function sets the path used by the specified engine for locating template files.

Parameter	Description
hEngine	Engine handle; should be obtained using call to DSEngineOpen
pszTemplatePath	Pointer to null-terminated string containing path of template files which may need to be accessed in creating engine programs for high-level functions

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the path could not be set. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Template files are source code files in the native language of the DSP Engine, and are used as the basis for creating executable engine programs. Depending on the DSP Engine, high-level functions such as DSAcquireWvfrmFile, DSGenerateWvfrmFile, and DSExecEngineFunction may be implemented using this technique.

C/C++ Example

```
DSSetEngineTemplatePath(hEngine, "c:\\dspower\\syslib");
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

section 4.5 Acquiring Data Using DirectCore Software

section 7.3 Executing Functions in DSP Engines

DSSetEngineWaveformPath

UINT DSSetEngineWaveformPath (hEngine, pszWaveformPath)

HENGINE hEngine	// engine handle
LPCSTR pszWaveformPath	// path for engine waveform files

The DSSetEngineWaveformPath function sets the path used by the specified engine for locating waveform files.

Parameter	Description
hEngine	Engine handle; should be obtained using call to DSEngineOpen
pszWaveformPath	Pointer to null-terminated string containing path of waveform files which might be accessed by the DSP Engine

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid, or the path could not be set. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

Waveform files can be in Hypersignal (.tim, .frq, etc.), MATLAB (.mat), Microsoft (.wav), or various acoustic/audio waveform file formats. For more information, see chapter 4, Waveform File Formats.

C/C++ Example

```
DSSetEngineWaveformPath(hEngine, "c:\\hsmacro\\waveform");
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

section 4.5 Acquiring Data Using DirectCore Software

section 7.3 Executing Functions in DSP Engines

DSShowEngMgrStatusWindow

void DSShowEngMgrStatusWindow ( )

The DSShowEngMgrStatusWindow function makes the Engine Manager error/status listbox window visible, if not already visible. The window's current contents and text display are not altered.

 Copyright © Signalogic 1994-2013

Parameter	Description

None

Returns

No return value.

Comments

At least one call to DSAddEngMgrStatusLine must occur before the error/status listbox is created.

By default, the error/status listbox is not visible. C/C++, Visual Basic, MATLAB, and LabView programs using Engine Manager (part of DirectCore and DirectCore software) must call DSShowEngMgrStatusWindow at some point to make the listbox visible.

C/C++ Example

```
DSAddEngMgrStatusLine("add this line before showing window");
DSShowEngMgrStatusWindow();
```

See Also

DSAddEngMgrStatusLine

DSHideEngMgrStatusWindow

Related Users Guide Topics

section 7.5.1 Internal Error Handling in Engine Manager

section 7.5.2 Engine Error Handling

section 7.5.3 Error/Status Window in Engine Manager

DSUnregisterEngineMsgWnd

UINT DSUnregisterEngineMsgWnd (hEngine, uMsgType)

```
HENGINE  hEngine     // engine handle
UINT  uMsgType       // message type(s)
```

The DSUnregisterEngineMsgWnd function unregisters the current window(s) associated with one or more types of callback messages, after which the specified callback message types are no longer sent to the window(s). A combination of message types can be specified using the pre-defined constants listed below.

Parameter	Description
hEngine	Engine handle which was previously obtained using call to DSEngineOpen
uMsgType	One or more message types for which the current registered window(s) should be unregistered, as defined below:

Value	Meaning
DS_REMW_SETENGINESTATUSMSG	unregister window currently receiving engine status messages
DS_REMW_SETPROGSTATUSMSG	unregister window currently receiving engine program and function status messages
DS_REMW_SETDSPDATARDYMSG	unregister window currently receiving buffer ready and flag ready messages
DS_REMW_SETSTATISTICSMSG	unregister window currently receiving statistic and parameter update messages
DS_REMW_SETBUFNUMMSG	unregister window currently receiving buffer number update messages
DS_REMW_SETLEVELMSG	unregister window currently receiving data level messages
DS_REMW_SETALLMSG	unregister all windows currently receiving any callback messages

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the engine handle was invalid or one or more message types could not be unregistered. For further error checking, use the DSGetEngineErrorStatus and DSGetEngMgrErrorStatus routines.

Comments

See section 6.1, Library Messages, for information about callback messages which can be received by application programs. The enmgr.h include file supplied with DirectCore software lists all callback message types.

C/C++ Example

```
DSUnregisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG | DS_REMW_SETDSPDATARDYMSG);
```

See Also

DSRegisterEngineMsgWnd

DSWaitForBuffer

DSWaitForFlag

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

2.2.4 Hardware Manager Functions

Below are listed functions in the Hardware Manager API. For information on how to include Hardware Manager (hwmgr.dll or hwmgr32.dll) in C/C++, Visual Basic, MATLAB, and LabView projects, see Signalogic Software Users Guide section 8.1, Hardware Manager Access.

Hardware Manager Functions

DSGetHWMgrCurVal

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrEntryIndex

DSGetHWMgrErrorStatus

DSGetHWMgrNumEntries

DSGetHWMgrVar

DSReadHWSetupFile

DSSetHWMgrCurVal

DSSetHWMgrVar

DSShowHardwareSelectorDlg

DSWriteHWSetupFile

DSGetHWMgrCurVal

UINT DSGetHWMgrCurVal (pszCardDesignator)

```
LPSTR  pszCardDesignator      // pointer to card designator string
```

The DSGetHWMgrCurVal function retrieves the card designator for the current hardware value selected in the Hardware Selector.

Parameter	Description
pszCardDesignator	Pointer to string variable which will contain a null-terminated card designator string upon return

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the card designator string for the current hardware value in the Hardware Selector could not be retrieved. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The string variable pointed to by pszCardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although card designators are normally no longer than 10-12 characters. For more information about card designators, see Card Designator Field in Hardware Setup File, in Signalogic Software Users Guide section 8.2.2, DSP/Analog Hardware Configuration.

For more information about setting the current hardware value (including the current Card Designator field) in the Hardware Selector, see Signalogic Software Users Guide section 8.2.1, Selecting DSP/Analog Hardware Types.

For more information about hardware values, see Signalogic Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSGetHWMgrCurVal function call being used to retrieve the current Card Designator from the Hardware Selector, and then using this value to assign a card handle.

```
char szCardDesignator[50];
int nErrorType;
HCARD hCard;

if (!DSGetHWMgrCurVal(szCardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else hCard = DSAssignCard(hEngine, szCardDesignator, NULL, NULL, NULL);
```

See Also

DSSetHWMgrCurVal

DSAssignCard

DSGetHWMgrEntryIndex

DSShowHardwareSelectorDlg

Related Users Guide Topics

8.2.1 Selecting DSP/Analog Hardware Types

8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

Card Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Card Designator Field in Hardware Setup File, in section 8.2.2, DSP/Analog Hardware Configuration

DSGetHWMgrEntry

UINT DSGetHWMgrEntry (wIndex, pHWList)

WORD wIndex	// index into Hardware Setup File
PHWLIST pHWList	// pointer to HWLISTENTRY structure

The DSGetHWMgrEntry function retrieves an entry from the Hardware Setup File, storing it in a HWLISTENTRY structure.

Parameter	Description
wIndex	Index specifying which Hardware Setup File entry to retrieve
pHWList	Pointer to HWLISTENTRY structure which will contain entry from Hardware Setup File

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the Hardware Setup File index was out of range. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The Hardware Setup File is initially read into global memory (allocated by Hardware Manager) from disk file upon the first instance of a call into Hardware Manager requiring data multicore CPU or DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

It is possible to obtain multicore CPU or DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry function with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, DSSetHWMgrEntry and DSWriteHWSetupFile calls should be made. Note that if the Hardware Selector is invoked by the operator, and multicore CPU or DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

For more information, see Signalogic Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntryIndex and DSGetHWMgrEntry calls to obtain Hardware Setup File information:

```
int nErrorType;
WORD wIndex;
HWLISTENTRY HWListEntry;

_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
wIndex = DSGetHWMgrEntryIndex("IIC31");
if (!wIndex || !DSGetHWMgrEntry(&HWListEntry, wIndex)) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that the HWListEntry structure is initialized to zero before using, causing most structure elements to be set their default state. Note also that DSGetHWMgrEntryIndex can return NULL if the Hardware Setup File entry specified by this function's pszCardDesignator input parameter is not found.

See Also

HWLISTENTRY

DSGetHWMgrEntryIndex

DSGetHWMgrNumEntries

DSSetHWMgrEntry

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSGetHWMgrEntryIndex

UINT DSGetHWMgrEntryIndex (pszCardDesignator)

LPCSTR pszCardDesignator // pointer to card designator string

The DSGetHWMgrEntryIndex function retrieves the index of the Hardware Setup File entry matching the specified Card Designator.

Parameter	Description
pszCardDesignator	pointer to null-terminated card designator string for which to find a Hardware Setup File index

Returns

The return value is NULL if the specified card designator string does not match any of the entries in the Hardware Setup File, or if there is a file-access problem. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Index values returned by DSGetHWMgrEntryIndex can range from 0 to the current total number of Hardware Setup File entries – 1, inclusively; the upper value can be determined by calling DSGetHWMgrNumEntries.

The DSGetHWMgrEntryIndex function can be combined with DSGetHWMgrEntry and DSSetHWMgrEntry to modify multicore CPU or DSP/analog hardware configuration data in the Hardware Setup File. The Reference Guide descriptions for the latter two functions contain comments and detailed source code examples which provide more information about this process.

For more information about the arrangement, contents, and access of the Hardware Setup File, see Signalogic Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntryIndex function to look for the index of an entry in the Hardware Setup File matching a particular Card Designator string:

```
int nErrorType;
WORD wIndex;

wIndex = DSGetHWMgrEntryIndex("ARI3210");
if (!wIndex) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

Note that DSGetHWMgrEntryIndex returns NULL if the Hardware Setup File entry specified by the function's Card Designator string input parameter is not found. For more complete C/C++ source code examples, see Reference Guide descriptions for DSGetHWMgrEntry and DSSetHWMgrEntry.

See Also

DSGetHWMgrNumEntries

DSGetHWMgrEntry

DSSetHWMgrEntry

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWMgrErrorStatus

UINT DSGetHWMgrErrorStatus ( )

The DSGetHWMgrErrorStatus function retrieves the current base-level error status for the Hardware Manager library (hwmgr.dll or hwmgr32.dll).

Parameter	Description
None	

Returns

The return value indicates the current Hardware Manager library error condition, if one exists; otherwise, the return value is NULL. See C/C++ source code example below.

Comments

The Hardware Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into Hardware Manager fail, the DSGetHWMgrErrorStatus call should be invoked.

A single, base-level current error condition variable is maintained internally in the Hardware Manager library. If an error condition occurs but is not retrieved, and further calls are made which cause *another* base-level library error condition, DSGetHWMgrErrorStatus returns only the most recent error condition.

Note that Engine Manager maintains an "error/status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of error conditions from all DirectCore library calls, including Hardware Manager.

For more information, see Signalogic Software Users Guide sections 8.4, Error Handling in Hardware Manager, and 8.4.1, Internal Error Handling in Hardware Manager. See also section 7.5.3, Error/Status Window in Engine Manager.

C/C++ Example

Below is an example showing error investigation in case of failure of a DSGetHWMgrEntryIndex call:

```
int nErrorType, Index;

Index = DSGetHWMgrEntryIndex("IIC31");
if (!Index) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

See Also

DSGetEngMgrErrorStatus

DSGetEngineErrorStatus

DSGetHWLibErrorStatus

Related Users Guide Topics

section 8.4 Error Handling in Hardware Manager

section 8.4.1 Internal Error Handling in Hardware Manager

section 7.5.3 Error/Status Window in Engine Manager

DSGetHWMgrNumEntries

UINT DSGetHWMgrNumEntries ( ),

The DSGetHWMgrNumEntries function retrieves the total number of current entries in the Hardware Setup File.

Parameter	Description
None	

Returns

The return value is NULL if the Hardware Setup File has not yet been read or cannot be read into Hardware Manager memory, or there are no valid entries in the file. Otherwise, the return value indicates the number of entries found in the Hardware Setup File. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

The DSGetHWMgrNumEntries function can be combined with DSGetHWMgrEntryIndex, DSGetHWMgrEntry, and DSSetHWMgrEntry to modify multicore CPU or DSP/analog hardware configuration data in the Hardware Setup File. The Reference Guide descriptions for the latter two functions contain comments and detailed source code examples which provide more information about this process.

The DSGetHWMgrEntryIndex function can be used to retrieve the index of a Hardware Setup File entry matching a specified card designator string. This function returns values ranging from 0 to DSGetHWMgrNumEntries – 1, inclusively.

For more information about the arrangement, contents, and access of the Hardware Setup File, see Signalogic Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrNumEntries function to create a loop in which Hardware Setup File entries with certain configuration data are searched for:

```
int nErrorType;
UINT uNumEntries, i;
HWLISTENTRY HWListEntry;
WORD wMemBaseAddr;

uNumEntries = DSGetHWMgrNumEntries();
if (!uNumEntries) {

    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
else for (i=0; i<uNumEntries; i++) {  // loop through all entries

    _fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));

    if (!DSGetHWMgrEntry(&HWListEntry, i)) {  // get the entry
```

```
        nErrorType = DSGetHWLibErrorStatus();
        ...do error processing...
        break; // exit loop
    }
    else {

        sscanf(HWListEntry.MemBaseAddr, %x, &wMemBaseAddr);

        if (wMemBaseAddr >= 0xD000  &&  wMemBaseAddr <= 0xEFFF) {

            // process memory-mapped hardware in region D000H - EFFFH
        }
    }
}
```

For additional C/C++ source code examples, see Reference Guide descriptions for DSGetHWMgrEntry and DSSetHWMgrEntry.

See Also

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWMgrVar

UINT DSGetHWMgrVar (wVarIndex, pszCardDesignator)

WORD wVarIndex // index of hardware variable

LPSTR pszCardDesignator // pointer to card designator string

The DSGetHWMgrVar function retrieves the card designator string for the specified hardware variable in the Hardware Selector.

Parameter	Description
wVarIndex	Index of hardware variable for which to retrieve card designator string; should be one of the pre-defined hardware variable constants listed below:

Value	Meaning
RTE	Real-Time Engine
SA	Spectrum Analyzer
ACC	DSP Accelerator

DTR	Digital Tape Recorder
ACQ	Data Acquisition (A/D conversion, waveform file record)
CSG	Continuous Signal Generation (periodic function generation and arbitrary signal synthesis)
SR	Stimulus & Response
PB	Playback (waveform file output)
DO	Digital Oscilloscope
UD1	User-Defined 1
UD2	User-Defined 2
UD3	User-Defined 3
UD4	User-Defined 4
UD5	User-Defined 5

pszCardDesignator	Pointer to string variable which will contain a null-terminated card designator string upon return

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the hardware variable was out of range (greater than or equal to MAXCARDVARS), or the card designator string could not be retrieved for the specified hardware variable in the Hardware Selector. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Hardware variables appear in a subdialog box that is invoked from the Hardware Selector by pushing the button marked "Set Hardware Variables". Currently, this button is enabled only when the Hardware Selector is displayed from DSPower-Block Diagram software. However, hardware variables can still be set and retrieved at any time by user-defined programs using the DSGetHWMgrVar and DSSetHWMgrVar functions in Hardware Manager.

The string variable pointed to by pszCardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although card designators are normally no longer than 10-12 characters. For more information about card designators, see Card Designator Field in Hardware Setup File, in Signalogic Software Users Guide section 8.3.1, Hardware Setup File Fields.

For more information about hardware variables, see Signalogic Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSGetHWMgrVar function call being used to retrieve the card designator string for the data acquisition hardware variable from the Hardware Selector, and then using this value to assign a card handle.

```
char szCardDesignator[50];
int nErrorType;
HCARD hCard;

if (!DSGetHWMgrVar(ACQ, szCardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else hCard = DSAssignCard(hEngine, szCardDesignator, NULL, NULL, NULL);
```

See Also

DSSetHWMgrVar

DSGetHWMgrEntryIndex

DSShowHardwareSelectorDlg

DSGetHWMgrCurVal

DSSetHWMgrCurVal

DSAssignCard

Related Users Guide Topics

section 8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software Card Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSGetHWSetupFile

UINT DSReadHWSetupFile (pszSetupFile)

```
LPSTR  pszSetupFile          // pointer to returned value of hardware setup file
```

The DSGetHWSetupFile function retrieves the current Hardware Setup File for the calling application. Note that no actual setup file access is made (see DSReadHWSetupFile description).

Parameter	Description
pszSetupFile	pointer to memory area sufficient to receive null-terminated Hardware Setup File string, which will contain full path information

Returns

The return value is NULL if the pszSetupFile parameter is NULL.

Comments

The current Hardware Setup File in use by a calling application can be specified by making DSSetHWSetupFile or DSReadHWSetupFile calls.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

For more information, see Signalogic Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

See Also

DSReadHWSetupFile

DSSetHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSReadHWSetupFile

HGLOBAL DSReadHWSetupFile (pszSetupFile)

LPCSTR pszSetupFile // pointer to hardware setup file

The DSReadHWSetupFile function allocates global memory and reads HWLISTENTRY entries from the specified Hardware Setup File (containing multicore CPU or DSP/analog hardware configuration data) into this memory A NULL value for pszSetupFile causes the default “hwsetup.lst” setup file, or previous setup file specified by DSSetHWSetupFile, to be read.

Parameter	Description
pszSetupFile	Pointer to null-terminated Hardware Setup File string; this string should specify the location of the setup file, including path information if needed; without path information, the default location used by Hardware Manager (location of hwmgr.dll file or hwmgr32.dll) will be used

Returns

The return value is NULL if either the specified Hardware Setup File could not be located or a problem occurred allocating memory or reading the file. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below). Otherwise, the return value is the global memory handle to a series of HWLISTENTRY records containing multicore CPU or DSP/analog hardware configuration data found in the specified setup file.

Comments

If no initial call is made to DSReadHWSetupFile, the default Hardware Setup File is initially read into global memory from disk file upon the first instance of a call into Hardware Manager requiring multicore CPU or DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

Memory is allocated as each new entry is read from the Hardware Setup File. If at some point, memory cannot be increased, NULL is returned and a subsequent call to DSGetHWMgrErrorStatus will return DSCOULDNOTINCREASEHWLISTMEM.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

It is possible to obtain and change multicore CPU or DSP/analog hardware configuration data programmatically by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. In this case, a DSWriteHWSetupFile call should be made if multicore CPU or DSP/analog hardware configuration data has been changed, and needs to be saved. Alternatively, setup files may be altered directly using a standard text editor, outside of the Hardware Manager environment. Note that if the Hardware Selector is invoked by the operator, and multicore CPU or DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

If the DSReadHWSetupFile function is successful, the return value is the global memory handle to a series of HWLISTENTRY records containing multicore CPU or DSP/analog hardware configuration data. The number of records can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access HWLISTENTRY records directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

Note that the current hardware value setting, as well as hardware variable settings, are not stored in the Hardware Setup File. They are instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini), on an application-independent basis (separate sections inside the configuration file are used).

For more information, see Signalogic Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

C/C++ Example

The example below shows the DSReadHWSetupFile function being called:

```
HGLOBAL hHWList;
int nErrorType;
LPCSTR szSetupFile = NULL;

hHWList = DSReadHWSetupFile(szSetupFile);

if (!hHWList) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSWriteHWSetupFile

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrNumEntries

DSShowHardwareSelectorDlg

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSSetHWMgrCurVal

UINT DSSetHWMgrCurVal (pszCardDesignator)

LPCSTR pszCardDesignator // pointer to card designator string

The DSSetHWMgrCurVal function sets the current hardware value in the Hardware Selector to match the specified card designator.

Parameter	Description
pszCardDesignator	pointer to null-terminated string containing card designator

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating that the card designator string was invalid or no match could be found in the current Hardware Setup File entries. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source example below).

Comments

For more information about the current hardware value in the Hardware Selector, see Signalogic Software Users Guide section 8.2.1, Selecting DSP/Analog Hardware Types.

For more information about hardware values, see Signalogic Software Users Guide section 8.2.3, Hardware Values and Variables.

For more information about card designators, see Card Designator Field in Hardware Setup File, in Signalogic Software Users Guide section 8.3.1, Hardware Setup File Fields.

C/C++ Example

The example below shows a DSSetHWMgrCurVal function call being used to set the current card designator prior to displaying the Hardware Selector dialog box:

```
char szCardDesignator[50], szUserCardDesignator[50];
int nErrorType;

lstrcpy(szCardDesignator, "TBS56-B");

if (!DSSetHWMgrCurVal(szCardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
else {

    DSShowHardwareSelectorDlg(hwnd, szUserCardDesignator); // hwnd is owner

    if (lstrcmpi(szCardDesignator, szUserCardDesignator) != 0) {

        // user has made a different choice; process...
    }
}
```

Note that the DSShowHardwareSelectorDlg call also returns a card designator. In the above example, if the user did not specify a new choice while the Hardware Selector was displayed, or pushed the "Cancel" button, then szCardDesignator and szUserCardDesignator would end up with the same value.

See Also

DSGetHWMgrCurVal

DSShowHardwareSelectorDlg

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.2.1 Selecting DSP/Analog Hardware Types

section 8.2.3 Hardware Values and Variables

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software Card Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSSetHWMgrEntry

UINT DSSetHWMgrEntry (wIndex, pHWList)

```
WORD  wIndex      // index into Hardware Setup File
PHWLIST  pHWList  // pointer to HWLISTENTRY structure
```

The DSSetHWMgrEntry function sets an entry in the Hardware Setup File, using information contained in a HWLISTENTRY structure.

Parameter	Description
wIndex	Index specifying which Hardware Setup File entry to set
pHWList	Pointer to HWLISTENTRY structure containing information with which to set the Hardware Setup File entry

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the wIndex parameter was out of range (greater than the current value returned by DSGetHWMgrNumEntries). For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

If wIndex is equal to the value returned by DSGetHWMgrNumEntries (i.e. wIndex is one more than the index of the last existing entry), the entry specified by pHWList will be added, rather than replacing an existing entry.

The Hardware Setup File is initially read into global memory (allocated by Hardware Manager) from disk file upon the first instance of a call into Hardware Manager requiring data multicore CPU or DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call.

It is possible to obtain and set multicore CPU or DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, a DSWriteHWSetupFile call should be made. Note that if the Hardware Selector is invoked by the operator, and multicore CPU or DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically.

For more information, see Signalogic Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

C/C++ Example

Below is an example of using the DSGetHWMgrEntry call to modify Hardware Setup File information, which has previously been retrieved using DSGetHWMgrEntryIndex and DSGetHWMgrEntry:

```
int nErrorType;
WORD wIndex;
HWLISTENTRY HWListEntry;

wIndex = DSGetHWMgrEntryIndex("IIC31");
if (!wIndex || !DSGetHWMgrEntry(&HWListEntry, wIndex)) {
```

```
    nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
else {

    // make changes to HWLISTENTRY information

    lstrcpy(HWListEntry.IOBaseAddr, "0x340");
    lstrcpy(HWListEntry.VoltageRanges, "1.4,1.8");
    HWListEntry.BusType = DS_AC_PCAT;

    if (!DSSetHWMgrEntry(&HWListEntry, wIndex)) {

        nErrorType = DSGetHWLibErrorStatus();
        ...do error processing...
    }
}
```

Note that the HWListEntry structure is initialized to zero before using, causing most structure elements to be set their default state. Note also that DSGetHWMgrEntryIndex can return NULL if the Hardware Setup File entry specified by the function's card designator string input parameter is not found.

See Also

HWLISTENTRY

DSGetHWMgrEntryIndex

DSGetHWMgrNumEntries

DSGetHWMgrEntry

DSReadHWSetupFile

DSWriteHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSSetHWMgrVar

UINT DSSetHWMgrVar (wVarIndex, pszCardDesignator)

WORD wVarIndex // index of hardware variable

LPCSTR pszCardDesignator // pointer to card designator string

The DSSetHWMgrVar function sets the card designator string for the specified hardware variable in the Hardware Selector.

Parameter	Description
wVarIndex	Index of hardware variable for which to set card designator string; should be one of the pre-defined hardware variable constants listed below:

Value	Meaning
RTE	Real-Time Engine
SA	Spectrum Analyzer
ACC	DSP Accelerator
DTR	Digital Tape Recorder
ACQ	Data Acquisition (A/D conversion, waveform file record)
CSG	Continuous Signal Generation (periodic function generation and arbitrary signal synthesis)
SR	Stimulus & Response
PB	Playback (waveform file output)
DO	Digital Oscilloscope
UD1	User-Defined 1
UD2	User-Defined 2
UD3	User-Defined 3
UD4	User-Defined 4
UD5	User-Defined 5

pszCardDesignator Pointer to string variable containing a null-terminated card designator string

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating the hardware variable was out of range (greater than or equal to MAXCARDVARS), indicating that the Card Designator string was invalid or no match could be found in the current Hardware Setup File entries. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Hardware variables appear in a subdialog box that is invoked from the Hardware Selector by pushing the button marked "Set Hardware Variables". Currently, this button is enabled only when the Hardware Selector is displayed from DSPower-Block Diagram software. However, hardware variables can still be set and retrieved at any time by user-defined programs using the DSGetHWMgrVar and DSSetHWMgrVar functions in Hardware Manager.

The string variable pointed to by pszCardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although card designators are normally no longer than 10-12 characters. For more information about card designators, see Card Designator Field in Hardware Setup File, in Signalogic Software Users Guide section 8.3.1, Hardware Setup File Fields.

For more information about hardware variables, see Signalogic Software Users Guide section 8.2.3, Hardware Values and Variables.

C/C++ Example

The example below shows a DSSetHWMgrVar function call being used to set the Card Designator string for the continuous signal generation hardware variable in the Hardware Selector:

```
int nErrorType;

if (!DSSetHWMgrVar(CSG, "NAT32C")) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSGetHWMgrVar

DSShowHardwareSelectorDlg

DSGetHWMgrCurVal

DSSetHWMgrCurVal

DSGetHWMgrEntryIndex

Related Users Guide Topics

section 8.2.3 Hardware Values and Variables

Card Designator, in section 8.2.2, DSP/Analog Hardware Configuration

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

Example C/C++ Code: Initializing DSP/Analog Hardware, in section 4.2.4, Example Code: DirectCore Software

DSSetHWSetupFile

UINT DSReadHWSetupFile (pszSetupFile)

```
LPCSTR  pszSetupFile        // pointer to hardware setup file
```

The DSSetHWSetupFile function sets the current Hardware Setup File for the calling application. Note that no actual setup file access is made (see DSReadHWSetupFile description).

Parameter	Description
pszSetupFile	pointer to null-terminated Hardware Setup File string; this string should specify the location of the setup file, including path information if needed; without path information, the default location used by Hardware Manager (location of hwmgr.dll file or hwmgr32.dll) will be used

Returns

The return value is NULL if the pszSetupFile parameter is NULL.

Comments

The current Hardware Setup File in use by a calling application can also be specified by making a DSReadHWSetupFile call.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

For more information, see Signalogic Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

See Also

DSGetHWSetupFile

DSReadHWSetupFile

DSWriteHWSetupFile

 Copyright © Signalogic 1994-2013

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

DSShowHardwareSelectorDlg

long DSShowHardwareSelectorDlg (hwnd, pszCardDesignator)

```
HWND  hwnd                  // handle of owner window
LPSTR  pszCardDesignator    // pointer to card designator string
```

The DSShowHardwareSelectorDlg function activates and displays the Hardware Selector dialog box.

Parameter	Description
hwnd	handle of owner window for Hardware Selector dialog box, if any
pszCardDesignator	pointer to string variable which will contain a null-terminated card designator string upon return

Returns

The return value is –1 if the Hardware Selector cannot be displayed, or if a problem occurs reading the Hardware Setup File. Otherwise, the low word of the return value contains the result of a DialogBoxParam call, and the high word contains a constant indicating whether configuration data for one or more hardware values was changed, or any hardware variables were changed, or both.

Typically, the low word of the return value will contain the identifier of the button pushed to close the dialog box, usually IDOK or IDCANCEL. The high word of the return value will have bit 0 set if the configuration data for a hardware value was changed, and bit 1 set if any hardware variables were changed. For further error checking if a –1 value should be returned, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

If not already read, the Hardware Setup File is read into global memory (allocated by Hardware Manager) from disk file when the DSShowHardwareSelectorDlg call is invoked.. If multicore CPU or DSP/analog hardware configuration data for one or more hardware values is changed by the operator while the Hardware Selector dialog box is displayed (and subsequently followed by pushing the "OK" button), a DSWriteHWSetupFile call is executed automatically before DSShowHardwareSelectorDlg returns.

It is also possible to obtain and set multicore CPU or DSP/analog hardware configuration data from Hardware Manager-allocated global memory by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. If the information is modified and needs to be saved, a DSWriteHWSetupFile call should also be made

For more information about the arrangement, contents, and means of access of the Hardware Setup File, see Signalogic Software Users Guide sections 8.3, Hardware Setup File, 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure (in section 8.3.2).

The string variable pointed to by pszCardDesignator is assumed to allow enough space for a return value (no space validation of any kind is performed). At least 50 characters is suggested, although card designators are normally no longer than 10-12 characters. For more information about card designators, see Card Designator Field in Hardware Setup File, in Signalogic Software Users Guide section 8.3.1, Hardware Setup File Fields.

The DSShowHardwareSelectorDlg invokes a DialogBoxParam call with "HardwareSetup" as the resource name, and pszCardDesignator as the extra parameter information.

C/C++ Example

Below is an example showing the DSShowHardwareSelectorDlg call being used to prepare for a Hardware Library DSAssignCard call:

```
char szCardDesignator[50];
HENGINE hEngine;

if (DSShowHardwareSelectorDlg(NULL, szCardDesignator) != IDCANCEL) {

    // assign a card handle:  engine, card designator, bus type, IO & mem base addrs
    hCard = DSAssignCard(hEngine, szCardDesignator, NULL, NULL, NULL);

    // initialize the card; make sure it's installed and responded
    if (!DSInitCard(hCard)) {

        itoa(DSGetEngineErrorStatus(hEngine), tmpstr, 10);
        MessageBox(NULL, tmpstr, "Test Prog:  DSInitCard failed", MB_OK);
    }
}
```

In the above example, DSShowHardwareSelectorDlg is used to obtain a valid card designator string for use in a DSAssignCard call. In this case, the software operator makes an onscreen choice using the hardware selection listbox inside the Hardware Selector. Note that the above example assumes that DSEngineOpen was called at some point to produce a valid engine handle for use in the DSAssignCard call.

See Also

DSGetHWMgrEntry

DSSetHWMgrEntry

DSReadHWSetupFile

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in this section 8.3.2

Card Designator Field in Hardware Setup File, in section 8.3.1, Hardware Setup File Fields

DSWriteHWSetupFile

UINT DSWriteHWSetupFile (hHWList)

HGLOBAL hHWList	// handle to multicore CPU or DSP/analog configuration data inside previously opened Hardware Setup File

The DSWriteHWSetupFile function writes current HWLISTENTRY records (containing multicore CPU or DSP/analog hardware configuration data) in the global memory area specified by hHWList to its associated Hardware Setup File (previously specified in calls to DSReadHWSetupFile or DSSetSetupFile).

 Copyright © Signalogic 1994-2013

Parameter	Description
hHWList	Handle to global memory area containing multicore CPU or DSP/analog hardware configuration data, maintained as series of HWLISTENTRY records

Returns

The return value is TRUE if successful. Otherwise, the return value is NULL, indicating a problem in accessing the global memory area or its associated Hardware Setup File. For further error checking, use the DSGetHWMgrErrorStatus routine (see C/C++ source code example below).

Comments

Note that if the hHWList parameter is given as NULL, then Hardware Manager determines the correct value assuming the calling application is the current Windows running task.

Currently, the default Hardware Setup File is named hwsetup.lst, and the default location of this file is the location of Hardware Manager (hwmgr.dll or hwmgr32.dll).

Hardware Manager keeps track of different setup files on an application-specific basis. Different applications may use different setup files or the same setup file. Also, different setup files may be open and updated at the same time. The current maximum number of applications which may be using Hardware Manager at one time is eight (8).

It is possible to obtain and change multicore CPU or DSP/analog hardware configuration data by calling the DSGetHWMgrEntry and DSSetHWMgrEntry functions with a pointer to a HWLISTENTRY structure. In this case, writing the Hardware Setup File using DSWriteHWSetupFile should be done if multicore CPU or DSP/analog hardware configuration data has been changed, and needs to be saved. Alternatively, setup files may be altered directly using a standard text editor, outside of the Hardware Manager environment. Note that if the Hardware Selector is invoked by the operator, and multicore CPU or DSP/analog hardware configuration data subsequently changed followed by pushing the "OK" button, a DSWriteHWSetupFile call is executed automatically.

The hHWList parameter, which may be a return value from a previous call to the DSReadHWSetupFile function is a global memory handle to a series of HWLISTENTRY records containing multicore CPU or DSP/analog hardware configuration data. The number of records can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access HWLISTENTRY records directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

Note that the current hardware value setting, as well as hardware variable settings, are not stored in the Hardware Setup File. They are instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini), on an application-independent basis (separate sections inside the configuration file are used).

For more information, see Signalogic Software Users Guide section 8.3, Hardware Setup File, section 8.3.2, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

C/C++ Example

The example below shows the DSWriteHWSetupFile function being called:

```
int nErrorType;

if (!DSWriteHWSetupFile(NULL)) {

    nErrorType = DSGetHWMgrErrorStatus();
    //  do error processing...
}
```

See Also

DSReadHWSetupFile

Copyrght © Signalogic 1994-2013

DSGetHWMgrEntry

DSSetHWMgrEntry

DSGetHWMgrNumEntries

DSShowHardwareSelectorDlg

Related Users Guide Topics

section 8.3 Hardware Setup File

section 8.3.1 Hardware Setup File Fields

section 8.3.2 Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2

 Copyright © Signalogic 1994-2013

2.3 About Core Lists

The currently selected core list is maintained internal to DirectCore software, and is accessed and set by numerous DirectCore APIs. Below is a list of functions that either implicitly or explicitly act on multiple cores, and their relationship to the currently selected core list.

Function	Action	Modifies Currently Selected Core List
DSGetCoreList	Return currently selected core list	N
DSGetProperty	Reads property from a core [1, 2]	N
DSInitCard	Initializes all cores	Y
DSLoadFileCard	Loads executable code to all cores	N
DSLoadFileCores	Loads executable code to specified cores [3]	N
DSResetCores	Resets specified cores [3]	N
DSRunCores	Runs specified cores [3]	N
DSReadMem	Reads memory from cores [1]	N
DSReadMemEx	Reads memory from specified cores [3]	N
DSSetCoreList	Sets currently selected core list [3]	Y
DSSetProperty	Sets property for one or more cores [1, 4]	N
DSWriteMem	Writes to memory of cores [1, 4.]	N
DSWriteMemEx	Writes to memory of specified cores [3, 4]	N

[1] Cores specified by currently selected core list.

[2] If multiple cores are currently selected, only the first core in the list (lowest bit position) is accessed.

[3] Cores specified by qwCoreList function parameter.

[4] If more than one core is specified, data written for each core is identical (“broadcast mode”). For DSWriteMemEx, the DS_WM_WRITEMULTIPLE attribute can be used to write different data for each core.

2.3.1 Abstraction of Cores

DirectCore software is designed with abstraction of hardware as a key objective. One aspect of the design is how cores are made visible and available to programmers. Here are some main points to keep in mind:

1) Hardware types may be mixed and matched, and each hardware (or card) adds to the number of available cores. Each hardware type must be assigned using the DSAssignCard and DSInitCard (or DSInitCores) APIs
2) Core lists have no "card boundaries" or other hardware-related distinction between cores. It is possible using the DSGetCardInfo API to determine which core positions in a core list belong to which card in the system
3) Core lists can be very long, up to 100s of cores, or more

2.3.2 API Core List Parameters

The qwCoreList parameter used in DirectCore APIs behaves in one of two (2) ways, depending on whether the DS_AC_CORELIST64 or DS_AC_CORELISTEXTENDED attributes were set when the card handle (hCard) was first assigned using a DSAssignCard API.

1) Legacy Core List. In this case, qwCoreList specifies cores as a bit field; i.e. each bit represents one (1) core. A core list value of 1 specifies core 0, a core list value of 0xff specifies cores 0-7, a core list value of 0xffffffff specifies cores 0-31, etc. Up to 64 cores can be specified. Since qwCoreList is defined as a QWORD, if a DWORD or other "shorter" parameter is used, then the core list is zero-extended to 64 bits.
2) Extended Core List. In this case, qwCoreList acts as a pointer to a structure containing a list of QWORDs that makes up the core list, with the first element of the structure a byte that contains the length of the list. Each QWORD uses the bit field convention as described in 1) above.

Below are some source code examples showing usage of both legacy core lists and extended core lists. The examples declare a "uNumCores" value representing the total number of available cores, then manipulate core lists in various ways – set all bits, set just one bit, clear a pattern of bits -- then write data to cores specified in the list.

Legacy Core List Example 1

```
// Set all bits in a core list for a 32-core system.  In this example, dwCoreList is sign-extended to a QWORD

UINT uNumCores = 32;
DWORD dwCoreList = pow(2, uNumCores) – 1;
DWORD dwDeviceMemAddr = 0x800;
UINT uBitWidth = DSGetWordLength(hCard);
DWORD dwVal = 0x12345678;
UINT uStatus;

uStatus = DSWriteMemEx(hCard, DS_RM_L2, dwDeviceMemAddr, uBitWidth, dwVal, 1, dwCoreList);


if (!uStatus) {

    // do error processing…
```

Legacy Core List Example 2

```
// set one bit in a core list for a 48-core system.  In this example core list is a QWORD, allowing up to 64 cores to be specified

UINT uNumCores = 48;
QWORD qwCoreList = 1 << 38; // set bit 37th bit
DWORD dwDeviceMemAddr = 0x800;
UINT uBitWidth = DSGetWordLength(hCard);
DWORD dwVal = 0x12345678;
UINT uStatus;

uStatus = DSWriteMemEx(hCard, DS_RM_L2, dwDeviceMemAddr, uBitWidth, dwVal, 1, qwCoreList);
```

```
if (!uStatus) {

    // do error processing…
```

Extended Core List Example 1

```
// Set all bits in a core list for a 128-core system.  Note that two (2) QWORDS are required

/* Note that this extended core list example assumes a DSAssignCard call was made previously (typically during code initialization code),
using the DS_AC_CORELISTEXTENDED attribute, for instance:

     hCard = DSAssignCard( NULL, "HardwareName", NULL, NULL, DS_AC_CORELISTEXTENDED, FALSE, NULL);
*/

DWORD dwDeviceMemAddr = 0x800;
UINT uBitWidth = DSGetWordLength(hCard);
DWORD dwVal = 0x12345678;
UINT uNumCores = 128;  // total number of accelerator cores
UINT uNumQwords = (uNumCores + MAXQWORDBITS – 1) / MAXQWORDBITS;
struct {
  BYTE count;
  QWORD* qwList;
} qwCoreList = { uNumQwords, calloc(uNumQwords, sizeof(QWORD)) };  // don't use calloc if this code is repeated many times…

for (i=uNumcores; i<uNumCores; i++) qwCoreList.qwList[i / MAXQWORDBITS] |= 1 << (i & (MAXQWORDBITS-1));

UINT uStatus = DSWriteMemEx(hCard, DS_RM_L2, dwDeviceMemAddr, uBitWidth, dwVal, 1, (QWORD)&qwCoreList);

if (!uStatus) {

    // do error processing…
```

Extended Core List Example 2

```
// Clear random bits in a core list for a 96-core system

/* Note that this extended core list example assumes a DSAssignCard call was made previously (typically during code initialization code),
using the DS_AC_CORELISTEXTENDED attribute, for instance:

     hCard = DSAssignCard( NULL, "HardwareName", NULL, NULL, DS_AC_CORELISTEXTENDED, FALSE, NULL);
*/

DWORD dwDeviceMemAddr = 0x800;
UINT uBitWidth = DSGetWordLength(hCard);
DWORD dwVal = 0x12345678;
UINT uNumCores = 96;  // total number of accelerator cores
UINT uNumQwords = (uNumCores + MAXQWORDBITS – 1) / MAXQWORDBITS;
struct {
  BYTE count;
  QWORD* qwList;
} qwCoreList = { uNumQwords, calloc(uNumQwords, sizeof(QWORD)) };  // don't use calloc if this code is repeated many times…

for (i=70; i<=75; i++) qwCoreList.qwList[i / MAXQWORDBITS] &= 1 << (i & (MAXQWORDBITS-1));

UINT uStatus = DSWriteMemEx(hCard, DS_RM_L2, dwDeviceMemAddr, uBitWidth, dwVal, 1, (QWORD)&qwCoreList);

if (!uStatus) {

    // do error processing…
```

2.4 RTAF Properties

Listed below are pre-defined constants which can be used in DSGetProperty and DSSetProperty calls in Hardware Library (DirectCore software), in order to control and communicate with multicore CPU or DSP code executing in conjunction with

1. Arbitrary, user-defined software running on the multicore CPU or DSP hardware. This includes any superset of the "minimal architecture" shown in Figure 1-3, where host application software makes DirectCore API calls to control and transfer data to/from multicore CPU or DSP software
2. Multicore CPU or DSP software auto-generated by the CIM accelerator framework
3. Hypersignal acquisition, generation, instrument, and file-based acceleration functions

DSGetProperty and DSSetProperty calls, as well as the DSReadMem and DSWriteMem calls, can be invoked either while the DSP is executing or in a static state.

DSPROP_ADCOUNT

DSPROP_CARDCLASS

DSPROP_CARDSUBCLASS

DSPROP_BUFLEN

DSPROP_BUFNUM

DSPROP_BUSYFLG

DSPROP_CHANLIST

DSPROP_CODECCTRL1

DSPROP_CODECCTRL2

DSPROP_COUPLIST

DSPROP_DACOUNT

DSPROP_DUPFLG

DSPROP_FILTADDR1

DSPROP_FILTADDR2

DSPROP_FILTLEN1

DSPROP_FILTLEN2

DSPROP_FILTTYPE1

DSPROP_FILTTYPE2

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSPROP_FFTLEN

DSPROP_FFTORD

DSPROP_FRMSIZ

DSPROP_FSMODE

DSPROP_FSVALUE

DSPROP_GAINLIST

DSPROP_HOSTBUFNUM

DSPROP_IOMODULE

Copyright © Signalogic 1994-2013

DSPROP_LOGCOEFFA
DSPROP_LOGCOEFFB
DSPROP_LOGFLG1
DSPROP_LOGFLG2
DSPROP_MAGDATAADDR
DSPROP_MAXVAL
DSPROP_MAXVAL1
DSPROP_MAXVAL2
DSPROP_MAXVALIMAG
DSPROP_MAXVALREAL
DSPROP_MINVAL
DSPROP_MONITORFLG
DSPROP_NUMCHAN
DSPROP_OFFSETIN
DSPROP_OFFSETOUT
DSPROP_OPMODE
DSPROP_OVERFLOW
DSPROP_PHZREQ
DSPROP_PWRCOEFFA
DSPROP_PWRCOEFFB
DSPROP_PWRFLG1
DSPROP_PWRFLG2
DSPROP_RIDATAADDR
DSPROP_RIFLG
DSPROP_SCALEIN
DSPROP_SCALEOUT
DSPROP_SERIALCTRL
DSPROP_STMBUFLEN
DSPROP_STMDATAADDR
DSPROP_TIMDATAADDR
DSPROP_TRIGCHANLIST
DSPROP_TRIGLEVEL
DSPROP_WINSCL
DSPROP_WINDATAADDR
DSPROP_XFERFLG1
DSPROP_XFERFLG2

DSPROP_ADCOUNT

Description

Acquisition delay counter, in samples.

See Also

DSPROP_DACOUNT

DSGetProperty

DSSetProperty

DSPROP_CARDCLASS

Description

Card classification number used in RTAF code executing on multicore CPU or DSP core(s).

Comments

Multicore CPU or DSP/analog hardware types are classified by number. If hardware types are close related, with only minor differences such as analog I/O characteristics, DSPROP_CARDSUBCLASS may be used.

See Also

DSPROP_CARDSUBCLASS

DSGetProperty

DSSetProperty

DSPROP_CARDSUBCLASS

Description

Card subclassification number used in RTAF code executing on multicore CPU or DSP core(s).

Comments

Multicore CPU or DSP/analog hardware types are normally classified by the DSPROP_CARDCLASS property. If hardware types are close related, with only minor differences such as analog I/O characteristics, DSPROP_CARDSUBCLASS may be used.

See Also

DSPROP_CARDCLASS

DSGetProperty

DSSetProperty

DSPROP_BUFLEN

Description

Buffer length used in acquisition, generation, instrument, or file-based acceleration processing functions.

See Also

DSPROP_FRMSIZ

DSGetProperty

DSSetProperty

DSPROP_BUFNUM

Description

In RTAF code, the current buffer flag in continuous analog I/O functions which use a buffer-switching mechanism.

See Also

DSPROP_HOSTBUFNUM

DSGetProperty

DSSetProperty

DSPROP_BUSYFLG

Description

Multicore CPU or DSP code busy flag.

Comments

Used by file-based acceleration processing functions to indicate when function is busy or idle.

See Also

DSPROP_RIFLG

DSGetProperty

DSSetProperty

DSPROP_CHANLIST

Description

Analog input channel list.

Comments

Each trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3.. For multicore CPU or DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSPROP_COUPLIST

DSPROP_GAINLIST

DSPROP_TRIGCHANLIST

DSGetProperty

DSSetProperty

DSPROP_CODECCTRL1

Description

CODEC control word 1.

Comments

Used only by multicore CPU or DSP/analog hardware types with analog I/O CODEC devices which have special configuration settings or options not controllable by standard properties sent from the DSP Engine. Examples of non-standard settings include sin(x)/x filter enable/disable, μLaw vs. linear data format, bit-order reversal, etc.

See Also

DSPROP_CODECCTRL2

DSPROP_SERIALCTRL

DSPROP_FSMODE

DSGetProperty

DSSetProperty

DSPROP_CODECCTRL2

Description

CODEC control word 2.

Comments

Used only by multicore CPU or DSP/analog hardware types with analog I/O CODEC devices which have special configuration settings or options not controllable by standard properties sent from the DSP Engine. Examples of non-standard settings include sin(x)/x filter enable/disable, μLaw vs. linear data format, bit-order reversal, etc.

See Also

DSPROP_CODECCTRL1

DSPROP_SERIALCTRL

DSPROP_FSMODE

DSGetProperty

DSSetProperty

DSPROP_COUPLIST

Description

Analog input coupling list.

Comments

Each input trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3. Currently defined values include 0 = dc coupling, 1 = ac coupling, 2 = microphone input, 3 = auxiliary input. For multicore CPU or DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSPROP_CHANLIST

DSPROP_GAINLIST

DSPROP_TRIGCHANLIST

DSGetProperty

DSSetProperty

DSPROP_DACOUNT

Description

Stimulus output delay counter, in samples.

See Also

DSPROP_ADCOUNT

DSGetProperty

DSSetProperty

DSPROP_DUPFLG

Description

Duplicate trace flag.

Comments

Set if instrument function trace 2 should be duplicated from trace 1.

See Also

DSGetProperty

DSSetProperty

DSPROP_FILTADDR1

Description

Filter coefficient address for trace 1 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Filter coefficients are either in floating-point or quantized fixed-point form, depending on the DSP device in use. Onchip DSP memory is used, if possible, for optimal real-time performance.

See Also

DSPROP_FILTLEN1

DSPROP_FILTTYPE1

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTADDR2

Description

Filter coefficient address for trace 2 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Filter coefficients are either in floating-point or quantized fixed-point form, depending on the DSP device in use. Onchip DSP memory is used, if possible, for optimal real-time performance.

See Also

DSPROP_FILTLEN2

DSPROP_FILTTYPE2

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTLEN1

Description

Length of filter for trace 1 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Specifies number of taps for FIR filter, filter order for IIR filters. A value of 0 indicates real-time filtering for trace 1 is disabled.

See Also

DSPROP_FILTADDR1

DSPROP_FILTLEN1

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTLEN2

Description

Length of filter for trace 2 in acquisition, generation, and instrumentation functions if real-time filtering enabled.

Comments

Specifies number of taps for FIR filter, filter order for IIR filters. A value of zero indicates real-time filtering for trace 2 is disabled.

See Also

DSPROP_FILTADDR2

DSPROP_FILTTYPE2

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTTYPE1

Description

Type of filter for trace 1 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Specifies FIR filter, IIR filter modified direct form, or IIR filter balanced pole-zero form.

See Also

DSPROP_FILTADDR1

DSPROP_FILTLEN1

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTTYPE2

Description

Type of filter for trace 2 in acquisition, generation, and instrumentation functions if real-time filtering is enabled.

Comments

Specifies FIR filter, IIR filter modified direct form, or IIR filter balanced pole-zero form. A value of –1 indicates that filter is identical to filter 1.

See Also

DSPROP_FILTADDR2

DSPROP_FILTLEN2

DSPROP_FILTQUANT

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTQUANT

Description

Filter coefficient and delay storage element quantization, in bits, used in acquisition, generation, and instrumentation functions which allow real-time filtering

See Also

DSPROP_FILTADDR1

DSPROP_FILTLEN1

DSPROP_FILTTYPE1

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FILTUPDFLG

Description

Filter coefficient update flag used in acquisition, generation, and instrumentation functions which allow real-time filtering.

Comments

Indicates that filter coefficients have changed and should be updated. Used mostly by multicore CPU or DSP/analog hardware types which have onchip memory not accessible by host server software, which requires filter coefficients to be moved into onchip memory from external memory when an update has occurred.

See Also

DSPROP_FILTADDR1

DSPROP_FILTLEN1

DSPROP_FILTTYPE1

DSPROP_FILTUPDFLG

DSGetProperty

DSSetProperty

DSPROP_FFTLEN

Description

FFT length used in FFT processing functions and spectrum analyzer function.

See Also

DSPROP_FFTORD

DSGetProperty

DSSetProperty

DSPROP_FFTORD

Description

FFT order used in FFT processing functions and spectrum analyzer function.

See Also

DSPROP_FFTLEN

DSGetProperty

DSSetProperty

DSPROP_FRMSIZ

Description

Framesize, in samples.

Comments

Used in file-based acceleration processing for DSP and math functions. Not used to control data acquisition, generation, or instrumentation buffer lengths.

See Also

DSPROP_BUFLEN

DSGetProperty

DSSetProperty

DSPROP_FSMODE

Description

Sampling rate clock generator mode or control register value.

Comments

Value used to initialize counter, timer, clock generator, etc. on multicore CPU or DSP/analog hardware used to generate sampling frequency for analog I/O functions. This value is normally calculated by DSP Engine, which takes into account discrete value, timer wordlength, or other limitations of multicore CPU or DSP/analog hardware.

See Also

DSPROP_FSVALUE

DSGetProperty

DSSetProperty

DSPROP_FSVALUE

Description

Actual sampling rate value, in Hz.

Comments

Actual sampling frequency value being used after DSPROP_FSMODE value has been determined. DSPROP_FSMODE is normally calculated by the DSP Engine, which takes into account discrete value, timer wordlength, or other limitations of multicore CPU or DSP/analog hardware. Note that for multicore CPU or DSP/analog hardware with 16-bit memory width, this value may not correctly reflect sampling rates greater than 65535.

See Also

DSPROP_FSMODE

DSGetProperty

DSSetProperty

DSPROP_GAINLIST

Description

Analog input gain list.

Comments

Each trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3.. For multicore CPU or DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSPROP_CHANLIST

DSPROP_COUPLIST

DSPROP_TRIGCHANLIST

DSGetProperty

DSSetProperty

DSPROP_HOSTBUFNUM

Description

Current host software buffer flag in continuous analog I/O functions that use a buffer-switching mechanism.

See Also

DSPROP_BUFNUM

DSGetProperty

DSSetProperty

DSPROP_IOMODULE

Description

I/O module indicator.

Comments

Currently only used by limited number of TMS320C25 hardware types.

 Copyright © Signalogic 1994-2013

See Also

DSGetProperty

DSSetProperty

DSPROP_LOGCOEFFA

Description

Log magnitude scaling coefficient "a".

Comments

Used in spectrum analyzer function when log magnitude calculation is in effect. Applied in equation of form Ylog(x) = $a*\log_{10}[Ylin(x)] + b$.

See Also

DSPROP_LOGCOEFFB

DSPROP_LOGFLG1

DSPROP_LOGFLG2

DSGetProperty

DSSetProperty

DSPROP_LOGCOEFFB

Description

Log magnitude scaling coefficient "b".

Comments

Used in spectrum analyzer function when log magnitude calculation is in effect. Applied in equation of form Ylog(x) = $a*\log_{10}[Ylin(x)] + b$.

See Also

DSPROP_LOGCOEFFA

DSPROP_LOGFLG1

DSPROP_LOGFLG2

DSGetProperty

DSSetProperty

DSPROP_LOGFLG1

Description

Log magnitude calculation request flag for trace 1.

Comments

Used by spectrum analyzer function.

See Also

DSPROP_LOGFLG2

DSPROP_LOGCOEFFA

DSPROP_LOGCOEFFB

DSGetProperty

DSSetProperty

DSPROP_LOGFLG2

Description

Log magnitude calculation request flag for trace 2.

Comments

Used by spectrum analyzer function.

See Also

DSPROP_LOGFLG1

DSPROP_LOGCOEFFA

DSPROP_LOGCOEFFB

DSGetProperty

DSSetProperty

DSPROP_MAGDATAADDR

Description

Frequency domain or wavelet domain magnitude/phase data base address.

Comments

For multicore CPU or DSP/analog hardware with linear memory, magnitude and phase values alternate locations. For hardware with vector memory, magnitude values are in X: memory and phase values in Y: memory.

See Also

DSPROP_RIDATAADDR

DSGetProperty

DSSetProperty

DSPROP_MAXVAL

Description

Maximum absolute value found in processing buffer.

See Also

DSPROP_MAXVALIMAG

DSPROP_MAXVALREAL

DSPROP_MINVAL

DSGetProperty

DSSetProperty

DSPROP_MAXVAL1

Description

Current maximum absolute amplitude value for trace 1 in data acquisition functions.

See Also

DSPROP_MAXVAL

DSPROP_MAXVAL2

DSGetProperty

DSSetProperty

DSPROP_MAXVAL2

Description

Current maximum absolute amplitude value for trace 2 in data acquisition function.

See Also

DSPROP_MAXVAL

DSPROP_MAXVAL1

DSGetProperty

DSSetProperty

DSPROP_MAXVALIMAG

Description

Maximum absolute value of imaginary component of complex data found in processing buffer.

See Also

DSPROP_MAXVAL

DSGetProperty

DSSetProperty

DSPROP_MAXVALREAL

Description

Maximum absolute value of real component of complex data found in processing buffer.

See Also

DSPROP_MAXVAL

DSGetProperty

DSSetProperty

DSPROP_MINVAL

Description

Minimum value found in acquisition or processing buffer.

See Also

DSPROP_MAXVAL

DSGetProperty

DSSetProperty

DSPROP_MONITORFLG

Description

Monitor mode enable/disable flag.

Comments

Used to control "monitor mode" in analog input functions. Enabling monitor mode causes analog input samples to looped back to analog output, with a one-sample delay, both before data acquisition is started, and during. If stimulus & response operation is in effect, input loopback automatically stops when stimulus output begins.

See Also

DSGetProperty

DSSetProperty

DSPROP_NUMCHAN

Description

Number of channels in analog I/O function.

See Also

DSPROP_CHANLIST

DSGetProperty

DSSetProperty

DSPROP_OFFSETIN

Description

Analog input data digital offset.

Comments

Specified in A/D samples. Value can be positive or negative and is added to analog input samples after digital scaling is applied (see DSPROP_SCALEIN).

See Also

DSPROP_SCALEIN

DSGetProperty

DSSetProperty

DSPROP_OFFSETOUT

Description

Analog output data digital offset.

Comments

Specified in D/A samples. Value can be positive or negative and is added to analog output samples after digital scaling is applied (see DSPROP_SCALEOUT).

See Also

DSPROP_SCALEOUT

DSGetProperty

DSSetProperty

DSPROP_OPMODE

Description

Operating mode.

Comments

Specifies processing or analog I/O function operating mode used in RTAF software.

See Also

DSPROP_CARDCLASS

DSGetProperty

DSSetProperty

DSPROP_OVERFLOW

Description

Overflow flag.

Comments

Indicates to host software when overflow has occurred in processing or instrumentation function. Active only for multicore CPU or DSP/analog hardware with fixed-point DSP devices.

See Also

DSGetProperty

DSSetProperty

DSPROP_PHZREQ

Description

Phase data calculation request flag.

Comments

Used in spectrum analyzer function.

See Also

DSPROP_LOGFLG1

DSPROP_LOGFLG2

DSGetProperty

DSSetProperty

DSPROP_PWRCOEFFA

Description

Coefficient "a" for power spectra calculation using exponential filter method.

Comments

Used in spectrum analyzer instrument function. Exponential filter is single-pole lowpass filter of form y[n] = a∗x[n] + b∗y[n−1].

See Also

DSPROP_PWRCOEFFB

DSPROP_PWRFLG1

DSPROP_PWRFLG2

DSGetProperty

DSSetProperty

DSPROP_PWRCOEFFB

Description

Coefficient "b" for power spectra calculation using exponential filter method.

Comments

Used in spectrum analyzer instrument function. Exponential filter is single-pole lowpass filter of form y[n] = a∗x[n] + b∗y[n−1].

See Also

DSPROP_PWRCOEFFA

DSPROP_PWRFLG1

DSPROP_PWRFLG2

DSGetProperty

DSSetProperty

DSPROP_PWRFLG1

Description

Power spectra calculation request flag for trace 1.

Comments

Used in spectrum analyzer instrument function.

See Also

DSPROP_PWRFLG2

DSPROP_PWRCOEFFA

DSPROP_PWRCOEFFB

DSGetProperty

DSSetProperty

DSPROP_PWRFLG2

Description

Power spectra calculation request flag for trace 2.

Comments

Used in spectrum analyzer instrument function.

See Also

DSPROP_PWRFLG1

DSPROP_PWRCOEFFA

DSPROP_PWRCOEFFB

DSGetProperty

DSSetProperty

DSPROP_RIDATAADDR

Description

Frequency domain or wavelet domain complex (real/imaginary) data base address.

Comments

For multicore CPU or DSP/analog hardware with linear memory, real and imaginary values alternate locations. For hardware with vector memory, real values are in X: memory and imaginary values in Y: memory.

See Also

DSPROP_MAGDATAADDR

DSGetProperty

DSSetProperty

DSPROP_RIFLG

Description

Real/imaginary data flag.

Comments

Used to indicate that DSP/math functions in RTAF software should expect and process real/imaginary data from host software data transfers.

See Also

DSPROP_RIDATAADDR

DSPROP_MAGDATAADDR

DSGetProperty

DSSetProperty

DSPROP_SCALEIN

Description

Analog input digital scale factor.

Comments

An integer value specified in "Q-10" notation, to allow both gain and attenuation. Typical effective scaling range is from 1/1024 to 256 is achieved by multiplying desired value by 1024. Applied before digital offset value is added (see DSPROP_OFFSETIN).

See Also

DSPROP_OFFSETIN

DSGetProperty

DSSetProperty

DSPROP_SCALEOUT

Description

Analog output data digital scaling factor.

Comments

An integer value specified in "Q-10" notation, to allow both gain and attenuation. Typical effective scaling range is from 1/1024 to 256 is achieved by multiplying desired value by 1024. Applied before digital offset value is added (see DSPROP_OFFSETOUT).

See Also

DSPROP_OFFSETOUT

DSGetProperty

DSSetProperty

DSPROP_SERIALCTRL

Description

Control word for DSP onchip serial port configuration.

Comments

multicore CPU or DSP/analog hardware types which use onchip serial ports to interface to external A/D or D/A converters most often require a fixed serial port control word value. However, some hardware types, most notably DSP32C and DSP3210, may allow some user-specified options in this value. In these cases, this value is active.

See Also

DSPROP_CODECCTRL1

DSPROP_CODECCTRL2

DSPROP_FSMODE

DSGetProperty

DSSetProperty

DSPROP_STMBUFLEN

Description

Stimulus data buffer length.

Comments

Stimulus data is output using dual-buffer switching mechanism; thus, total stimulus length is twice value of DSPROP_STMBUFLEN.

See Also

DSPROP_STMDATAADDR

DSGetProperty

DSSetProperty

DSPROP_STMDATAADDR

Description

Stimulus data base address in DSP memory.

Comments

For TMS320C2x and TMS320C5x multicore CPU or DSP/analog hardware, this address is in program memory. For all other hardware types, this address is in data memory.

See Also

DSPROP_STMBUFLEN

DSGetProperty

DSSetProperty

DSPROP_TIMDATAADDR

Description

Time domain data base address in DSP memory.

Comments

Used for both file-based acceleration processing and analog I/O functions.

See Also

DSPROP_WINDATAADDR

DSPROP_RIDATAADDR

DSGetProperty

DSSetProperty

DSPROP_TRIGCHANLIST

Description

Analog input trigger channel list.

Comments

Each trace occupies a 4-bit channel field in this value, starting with trace 0 in bits 0..3.. For multicore CPU or DSP/analog hardware types with 32-bit floating-point DSP devices, this normally means a maximum of 8 channels can be specified.

See Also

DSPROP_TRIGLEVEL

DSPROP_CHANLIST

DSGetProperty

DSSetProperty

DSPROP_TRIGLEVEL

Description

Analog input trigger level.

Comments

Specified in A/D samples; can be either positive or negative. A value of zero indicates "freerun" trigger mode.

See Also

DSPROP_TRIGCHANLIST

DSGetProperty

DSSetProperty

DSPROP_WINSCL

Description

Scaling factor compensating for specified FFT or wavelet transform window type, if any.

Comments

Used by spectrum analyzer function when calculating correct linear magnitude scale factors.

See Also

DSPROP_WINDATAADDR

DSPROP_LOGFLG1

DSPROP_LOGCOEFFA

DSGetProperty

DSSetProperty

DSPROP_WINDATAADDR

Description

Time domain window base address in DSP memory.

Comments

Used for FFT or other DSP processing function, and spectrum analyzer instrument function.

See Also

DSPROP_WINSCL

DSGetProperty

DSSetProperty

DSPROP_XFERFLG1

Description

Transfer function calculation request flag in spectrum analyzer function for trace 1.

See Also

DSPROP_XFERFLG2

DSGetProperty

DSSetProperty

DSPROP_XFERFLG2

Description

Transfer function calculation request flag for trace 2 in spectrum analyzer instrument function.

See Also

Copyrght © Signalogic 1994-2013

DSPROP_XFERFLG1

DSGetProperty

DSSetProperty

 Copyright © Signalogic 1994-2013

2.5 DSP Engine Functions

DSP Engine simulation functions have been further subdivided into the following categories:

- Hypersignal-Macro series software functions
- MATLAB functions

Within each category, functions are listed alphabetically. Note that with the online Help version of documentation, you can use the Browse buttons (">>" and "<<") to move forward and backwards between different function descriptions.

For a complete listing of all DirectCore software functions, see section 2, API Reference, above.

NOTE: In function descriptions listed below, knowledge of Hypersignal macro language and MATLAB .m program definition, variable declarations, and syntax is assumed.

 Copyright © Signalogic 1994-2013

3 Waveform File Formats

Overview

This section provides a list of the various file formats supported directly and indirectly by Signalogic software. There are three levels of waveform file format support:

1. Format is supported directly by software. Software functions either read format automatically by using a search based on waveform format filename extension, or require a waveform filename extension to be entered explicitly.

2. Format can be converted by utility function to a format supported directly by software. If waveform file format has a header, all values and attributes will be converted that have counterparts in the directly supported format.

Format can be "imported", using a general-purpose File Acquisition or similar utility to convert file data into a format directly supported by the software. In this case, if the waveform file format has a header, it is likely that some header values and attributes may not be converted fully, or lost.

Main Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

 Copyrght © Signalogic 1994-2013

3.1 Hypersignal Waveform File Format

Below is a quick reference for version 4.x and higher Hypersignal waveform file format. For additional, detailed information about individual values in the Hypersignal waveform file format, see Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format.

Table 1 Hypersignal Waveform File Format Quick Reference

Name	Word[1] Index	Size / Type
SHORT HEADER		
Maximum Amplitude	0	word
Framesize	1	word
Sampling Frequency remainder[2]	2	integer (0 to 32766)
FFT Order	3 (b7-b0)	byte
Amplitude Exponent	3 (b15-b8)	byte
Waveform Number of Elements (NE) remainder	4	integer (0 to 16383)
Frame Overlap	5	word
Data Type Attributes	6	
Time Domain Source	6 (b1-b0)	2 bits
Extended Header Attribute	6 (b2)	1 bit
Complex Attribute	6 (b3)	1 bit
LPC Order	6 (b8-b4)	5 bits
Window Type	6 (b11-b9)	3 bits
Amplitude Exponent	6 (b15-b12)	4 bits
Number of Channels	7 (b5-b0)	6 bits
Nyquist FFT pt. Attribute	7 (b6)	1 bit
Autoscale Attribute	7 (b7)	1 bit
User-Defined	7 (b15-b8)	byte
Sampling Frequency quotient[2]	8	integer (0 to 32767)
Waveform Number of Elements (NE) quotient	9	integer (0 to 32767)
EXTENDED HEADER		
Minimum Amplitude	10	word
Calibration Offset	11	single
Gain1	13	single
Gain2	15	single
Data Precision	17	word
Inp Level[3]	18	single
Start Freq	20	single
Extended Framesize[4]	22	doubleword
FFT Size	24	doubleword
Extended Frame Overlap[5]	26	doubleword
Floating-Point Amplitude	28	double

 Copyright © Signalogic 1994-2013

Transform Window Scale Value	32	single
Reserved for Signalogic Use	**34**	**14 words**
User-Defined	**48**	**16 words**
Start of Waveform Data	**64**	**NE integers**

Notes

[1] Unless otherwise mentioned, words are 16-bit unsigned integers ranging in value from 0..65535, integers are 16-bit values ranging from -32768..32767, doublewords are 32-bit unsigned integers ranging in value from 0..4294967295, singles are IEEE format 32-bit floating-point values, and doubles are IEEE format 64-bit floating-point values.

[2] See Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, for explanation on Sampling Frequency remainder, quotient and Waveform length remainder and quotient

[3] The QUANT field in earlier versions of Hypersignal has been removed. Its value is now contained as part of the multiplier in the Inp Level field. The new Inp Level value is calculated as 2**(QUANT-15)

[4] Valid only when Framesize field is zero.

[5] Valid only when Frame Overlap field is zero.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

3.2 Microsoft .WAV Waveform File Format

The "standard" Microsoft .WAV file format, unless modified by user-defined extension, is exclusively a time domain format. By standard, we mean .WAV files shipped with Windows 3.x and popular off-the-shelf multimedia products. The standard format contains a simple 44-byte header, which allocates no values for frequency domain information, wavelet domain information, or for other math/DSP analysis purposes. Furthermore, although the .WAV format is theoretically a subset of the general Microsoft RIFF format, and can thus be extended to contain user-defined header sections, experience at Signalogic has shown that most off-the-shelf software packages fail to ignore sections they do not understand, and either crash or process incorrect data.

Below is a quick reference for standard .WAV file formats. See the *Microsoft Windows Multimedia Programmer's Reference*, chapter 8, for detailed information on the .WAV format. Also, see the topic Hypersignal Tim2Wav and Wav2Tim Waveform File Conversion, in Signalogic Software System Guide section 1.2.1, Summary Of Hypersignal-Macro Series Software.

Table 2 Microsoft WAVE File Format Quick Reference

Name	Word[1] Index	Size / Type
SHORT HEADER		
'RIFF" section tag	0	4 bytes
Length of RIFF section	2	doubleword
'WAVE' format identifier	4	4 bytes
'fmt ' section tag	6	4 bytes
Length of fmt section	8	doubleword
PCM identifier	10	word
Number of Channels	11	integer
Sampling Frequency	12	doubleword
Average Bytes Per Second	14	doubleword
Block Alignment	16	integer
Precision (8-bit or 16-bit)	17	word
'data' section tag	18	4 bytes
Length of data section (NB)	20	doubleword
Start of Waveform Data	**22**	**NB/2 words**

Notes

[1] Unless otherwise mentioned, words are 16-bit unsigned integers ranging in value from 0..65535, integers are 16-bit values ranging from -32768..32767, doublewords are 32-bit unsigned integers ranging in value from 0..4294967295, singles are IEEE format 32-bit floating-point values, and doubles are IEEE format 64-bit floating-point values.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

Additional Notes About .WAV File Limitations

There are several limitations to the de facto standard 44-byte header .WAV format found in use with most multimedia programs. One of the most noticeable is the lack of maximum amplitude value, which causes .WAV files to be displayed and processed in Hypersignal and DSPower software as if they always contained full-scale values. Another is lack of frequency or wavelet domain information. These are severe limitations when working with Hypersignal or DSPower software, and it is strongly recommended that, at the earliest opportunity, you convert your .WAV files into Hypersignal .TIM format one of the following ways:

- Use the Wav2Tim conversion function either from the Hypersignal Macro Manger menu or as a DSPower-Block Diagram block.
- Use File Acquisition function, either from Hypersignal or as DSPower-Block Diagram block.
- Use the .WAV file as direct input into Hypersignal or DSPower-Block Diagram function, by specifying the .WAV extension explicitly in an input waveform filename field.
- MATLAB Waveform File Format

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

Copyrght © Signalogic 1994-2013

3.3 Acoustic/Audio Waveform File Formats

Mainly due to the influence of Hypersignal-Acoustic, information is available for some popular acoustic/audio analysis formats in the following sections.

Acoustic/Audio Waveform File Formats

 Copyright © Signalogic 1994-2013

3.3.1 Techron .TDS File Format

The .TDS file format is a frequency domain ASCII (text file) format used by Techron with their popular TEF20 analyzer system and SoundLab software. A tefrd.mac function is included with Hypersignal-Acoustic software, which can convert Techron .TDS files into Hypersignal .FRQ, .PHZ, and .REA/.IMA waveform files. These files can then be used directly in Hypersignal frequency domain functions and displays, including iFFT to produce equivalent time domain responses, and the equalize.mac function, to produce equalizing time domain responses.

The tefrd.mac function invokes the tefrd.exe utility, for which source code is included with Hypersignal-Acoustic software. Please see this source code for further information on the .TDS file format.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

Copyrght © Signalogic 1994-2013

3.3.2 Ariel SYSid File Format

The .DAT file format is a frequency domain ASCII (text file) format used by Ariel with their popular SYSid analyzer system and software. A sysidrd.mac function is included with Hypersignal-Acoustic software, which can convert Ariel .DAT files into Hypersignal .FRQ, .PHZ, and .REA/.IMA waveform files. These files can then be used directly in Hypersignal frequency domain functions and displays, including iFFT to produce equivalent time domain responses, and the equalize.mac function, to produce equalizing time domain responses.

The sysidrd.mac function invokes the sysidrd.exe utility, for which source code is included with Hypersignal-Acoustic software. Please see this source code for further information on SYSid file formats.

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

 Copyright © Signalogic 1994-2013

3.3.3 Micro Technology Unlimited .SF File Format

Micro Technology Unlimited (MTU) .SF format waveform files can be imported into Hypersignal-Macro series software or DSPower software formats using the File Acquisition function. Use the following settings:

- format set to 'BIN'
- sample width set to 4 (16-bit)
- bytes to discard set to 1024 for mono or stereo waveform input, or set to 1026 to import only channel B from a stereo waveform file
- number of samples to acquire set to 0 (all)
- sampling frequency set to appropriate value
- channel strip set to 0 for mono or stereo waveform input, or set to 1 to import either only channel A or channel B from a stereo waveform file
- number of channels set to 1 for mono acquisition, or 2 for stereo acquisition

General Waveform File Format Topics

Hypersignal Waveform File Format

Microsoft .WAV Waveform File Format

Additional Notes About .WAV File Limitations

MATLAB Waveform File Format

Acoustic/Audio Waveform File Formats

Techron .TDS File Format

Ariel SYSid File Format

Micro Technology Unlimited .SF File Format

Copyrght © Signalogic 1994-2013

4 Definitions

Overview

This section of the Signalogic DirectCore Software Reference Guide documents structures defined for use in application-specific C/C++ and MATLAB programs when calling functions in DSPower series DLLs. The actual (and current) definition of these structures can be found in the include files noted for each. Examples include hwlib.h, hwmgr.h, etc.

Structures Used in DSPower Series Software

HWLISTENTRY

CONVERSIONINFO

ERRORMSGINFO

DSPDATARDYMSGINFO

4.1 Structure Definitions

There are several structure definitions used by DSPower series software. These are typically found in include files such as enmgr.h, hwmgr.h, hwlib.h, and splib.h. The sections below describe these structure definitions.

4.1.1 HWLISTENTRY Structure

Below is a description of the HWLISTENTRY structure, which is used in Hardware Manager calls such as DSGetHWMgrEntry and DSSetHWMgrEntry. The current definition of this structure is located in the DirectCore include file **hwmgr.h**.

The description below is divided into the following sections:

- information maintained in the Hardware Setup File
- information maintained internally by Hardware Manager, and is not stored in the Hardware Setup File

Each element in the structure description below includes a brief comment. See also the notes after the structure definition, and Signalogic Software Users Guide section 8.3.1, Hardware Setup File Fields.

typedef struct {

// type	// element	// comment
DESCSTR	szMfg;	// manufacturer field
DESCSTR	szDescription;	// description field: name/model, DSP type, etc.
DESCSTR	szCardDesignator;	// card designator field
WORD	MajorDriverID;	// major multicore CPU or DSP/analog hardware driver ID field
WORD	MinorDriverID;	// minor multicore CPU or DSP/analog hardware driver ID field
char	szCodeFile[144];	// program (executable code) file (current program filename)
char	szCodeFileDef[144];	// default multicore CPU or DSP program (executable code) file
VALSTR	szIOBaseAddr;	// current I/O base address field (in hex)
VALSTR	szIOBaseAddrDef;	// default I/O base address
VALSTR	szMemBaseAddr;	// current memory base address field (in hex)
VALSTR	szMemBaseAddrDef;	// default base memory address
WORD	BusType;	// bus type field
WORD	BusTypeDef;	// default bus type
VALSTR	szProcClcok;	// core clock field (specified in MHz)
VALSTR	szProcClcokDef;	// default core clock
VALSTR	szVoltageRanges;	// voltage ranges field (should be entered as 'input, output')
VALSTR	szVoltageRangesDef;	// default voltage ranges
WORD	AnalClock;	// analog clock type field: 0 = external clock, 1 = fixed internal, 2 = prog internal
WORD	AnalClockDef;	// default analog clock type
VALSTR	szMemArch;	// memory architecture field; "L" = linear, "H" = modified Harvard, "V" = vector
VALSTR	szMemArchDef;	// default memory architecture
WORD	NumCores;	// number of cores field
WORD	NumCoresDef;	// default number of cores

// following parameters not stored in Hardware Setup File

// type	// element	// comment
WORD	CallClass;	// calls-allowed classification
WORD	Flags;	// internal information used to mark pending updates, deletes, etc.

} HWLISTENTRY;

typedef HWLISTENTRY* PHWLISTENTRY;

Notes

1) HWLISTENTRY structures should be completely initialized to zero before using, so default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
```

2) Supporting type definitions used above:

```
typedef char DESCSTR[50];
typedef char VALSTR[10];
```

Example

See Also

DSSetHWMgrEntry

DSGetHWMgrEntry

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

HWLISTENTRY Example

Below is an excerpt from the dscope.cpp source code example file included with DirectCore software. It shows a typical include file section which allows compilation of applications using the DirectCore software. Remember that referring to the prototypes inside the DirectCore DLL include files shown below will cause the necessary .lib (library) files to be needed in the application's project file at link-time; for more information on DirectCore files, see section 4.2.1, .LIB Files for DirectCore.

```
// typical application-specific include files
#include <windows.h>
#include <stdlib.h>
#include <math.h>

// DirectCore DLL include files
#include "enmgr.h"   // Engine Manager
#include "hwmgr.h"   // Hardware Manager
#include "hwlib.h"   // Hardware Library

// more Signalogic DLL include files, but used only for user-interface in demo
#include "toollib.h"
#include "helplib.h"
#include "minmax.h"  // local versions of min/max, if needed
```

Copyright © Signalogic 1994-2013

See Also

DSSetHWMgrEntry

DSGetHWMgrEntry

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

4.1.2 CONVERSIONINFO Structure

Below is a description of the CONVERSIONINFO structure, which is used in high-level DirectCore calls such as DSAcquireWvfrmFile and DSGenerateWvfrmFile. The current definition of this structure is located in the DirectCore include file **hwlib.h**.

The reprint below is partially divided into the following sections:

- basic acquisition/generation information
- real-time digital filter information
- trigger information
- stimulus & response (simultaneous input/output information)

Each element in the structure description below includes a brief comment. For more detailed information, see the additional explanations of some elements after the structure.

typedef struct {

// basic acquisition/generation information

// type	**// element**	**// comment**
char	szFilename[144]	// [path]filename of file being acquired/generated (see notes below)
WORD	NumChan	// number of channels
WORD	ChanList[MAXCHAN]	// channel list, maps physical channels to waveform traces (entries should not exceed NumChan)
float	Fs	// sampling rate (in Hz)
DWORD	NumSamples	// number of samples to acquire/generate
char	szGainList[41]	// string specifying gain list for input channels; can contain ',' and '-' other characters to specify lists and ranges (see notes below)
long	Offset	// offset added to data (in A/D counts, can be +/-)
BOOL	DALooping	// enables/disables continuous D/A looping; 0 or 1, valid only during generation (output)
BOOL	Monitor	// enables/disables D/A loopback when recording; 0 or 1, valid only during acquisition (input)
BOOL	FileSplit	// 1 enables A/D file-splitting for multichannel, 0 disables
BOOL	StoreOnAbort	// 1 causes current waveform data to be stored if a user- or program-induced abort occurs
DWORD	Frmsiz	// framesize (stored in waveform header; used for post-processing only)

// real-time digital filter information

// type	**// element**	**// brief description**
char	szFilter1[13]	// filename of real-time digital filter file; can contain '/' character
char	szFilter2[13]	// to allow dual-filter entry (filter files must be located on same path as waveform)

// trigger information

// type	**// element**	**// brief description**
WORD	TrigMode	// trigger mode (0 = internal, 1 = external)
long	TrigLevel	// trigger level (in A/D counts, can be +/-)
long	TrigDelay	// delay from trigger relative to data (in sec, can be ±)
WORD	wNumTrigChan	// number of trigger channels
WORD	TrigChanList[MAXCHAN]	// trigger channel list (entries should not exceed NumTrigChan

// stimulus & response information

// type	**// element**	**// brief description**
char	szStimName[13]	// name of stimulus output file; used only for DSStimRespMeasure function (must be located on same path as waveform)

WORD	StimMode	// controls stimulus usage in Stimulus/Response function
long	StimDelay	// delay from stimulus onset to start of record (+/- value)

} CONVERSIONINFO;

typedef CONVERSIONINFO* PCONVERSIONINFO;

Notes

1) The entire CONVERSIONINFO structure should be initialized to zero before using, so that default values can take effect for parameters not used. This can easily be accomplished using a call such as

_fmemset(&ConversionInfo, 0, sizeof(CONVERSIONINFO));

2) Previous value used in either DSEngineOpen or DSSetEngineWaveformPath call is used for waveform filenames, unless explicit path information is found in the filename above.

NOTE: For operation when a Hypersignal-Macro series program is the DSP Engine, szFilename above should not contain path information. The path should instead be set using the initial DSEngineOpen call, or a subsequent DSSetEngineWaveformPath call.

3) The szGainList parameter is a string because input (or output) gain and/or attenuation can be specified as a software digital scaling value, or as a hardware gain value in Volts or dB, depending on the DSP/analog hardware type. Also, entry can be channel-specific, or specified as a channel-range.

4) TrigDelay parameter not yet operational.

5) PCONVERSIONINFO can be used to declare pointers to CONVERSIONINFO structures. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in Signalogic Software Users Guide section 4.2.2, Example Code: DirectCore Software.

Example

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

Related Topics

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DirectCore Software

Creating Programs for DirectCore

Acquiring Data Using DirectCore Software

Using Engine Manager

Using Hardware Manager

CONVERSIONINFO Example

Below is a partial excerpt from the dtape.cpp example source code file included with DirectCore software showing typical setup and initialization of a CONVERSIONINFO structure for use in a DSAcquireWvfrmFile call. The dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine. One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see notes in section 5.1.2 above, CONVERSIONINFO Structure.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE | LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to acquire

// setup for one channel, init channel list (example is sequential)

pConvInfo->NumChan = 1;          // one channel
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz sampling rate
pConvInfo->NumSamples = 100000L;  // 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO
pConvInfo->NumTrigChan = 1;  // one trigger channel
pConvInfo->TrigChanList[0] = 0;  // freerun trigger

// start acquisition

DSAcquireWvfrmFile(hCard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

See Also

DSAcquireWvfrmFile

DSGenerateWvfrmFile

Related Topics

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DirectCore Software

Creating Programs for DirectCore

Acquiring Data Using DirectCore Software

Using Engine Manager

Using Hardware Manager

4.1.3 DSPDATARDYMSGINFO Structure

Below is a description of the DSPDATARDYMSGINFO structure, which pointed to by the lParam parameter of DSP Engine callback messages such as WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY sent by Engine Manager. The current definition of this structure is located in the DirectCore include file **enmgr.h**.

Each element in the structure description below includes a brief comment. See also the notes below, after the structure definition, and Signalogic Software Users Guide section Buffer Ready Messages from DSP Engines and Semaphore Messages from DSP Engines, under section 7.4, Registering Callback Messages from DSP Engines.

typedef struct {

// type	// element	// comment
WORD	wNum;	// buffer or flag number
DWORD	dwFlagValue;	// flag value
DWORD	dwFlagAddr;	// flag address
HGLOBAL	hCard;	// card handle

} DSPDATARDYMSGINFO;

typedef DSPDATARDYMSGINFO* PDSPDATARDYMSGINFO;

Notes

1) The buffer or flag number reflects the original wBufferNum or wFlagNum values in used in DSWaitForBuffer or DSWaitForFlag calls.

2) The lParam value of the WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY messages points to a DSPDATARDYMSGINFO structure.

3) The wParam value of the WM_DSPENGINE_BUFRDY and WM_DSPENGINE_FLGRDY messages is the handle of the engine to which the DSP card buffer or flag was assigned.

Example

See Also

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_FLGRDY

Related Users Guide Topics

Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

DSPDATARDYMSGINFO Example

Below is an excerpt from the dscope.cpp source code example file (included with DirectCore software) showing a WM_DSPENGINE_BUFRDY message being received:

```
case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int*)GlobalLock(hBuf);  // lock memory to be used
```

```
// transfer below illustrates two common type of DSP memory architectures

if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

    if (nCurBuf == 0)  status = DSReadMem(hCard, DS_RM_VECTOR_DATA_X, dwTimBufferBaseAddr, DS_RM_SIZE16, pBuf,
    wBuflen);

    else  status = DSReadMem(hCard, DS_RM_VECTOR_DATA_Y, dwTimBufferBaseAddr, DS_RM_SIZE16, pBuf, wBuflen);
}
else { // linear data memory

    status = DSReadMem(hCard, DS_RM_LINEAR_DATA_RT, TimBufferBaseAddr+nCurBuf*Buflen, DS_RM_SIZE16, pBuf, wBuflen);
}

GlobalUnlock(hBuf);  // let memory float

if (!status) MessageBox(NULL, "DSReadMem problem", "Test Prog", MB_OK);

// switch buffers and write new buffer number

nCurBuf ^= 1;
DSPutEngineVarMem(hCard, DSPROP_HOSTBUFNUM, nCurBuf);

// can we have another please, Sir..
DSWaitForBuffer(hCard, nCurBuf, NULL, DS_WFB_POLLED);
break;
```

Notes

1) A great deal of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the entire dscope.cpp file be examined before implementing code similar to the above example.

2) uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

3) hBuf is assumed to have been created via a GlobalAlloc call, using a sizeof parameter equal to 2*wBuflen. Memory pointers passed to DSReadMem and DSWriteMem calls *must not point to memory segments less than the product of the dwCount and DS_RM_SIZExx parameters* used in the corresponding DSReadMem or DSWriteMem calls. This product is in bytes.

See Also

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_FLGRDY

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

4.1.4 ERRORMSGINFO Structure

Below is a description of the ERRORMSGINFO structure, which is pointed to by the lParam parameter in DSP Engine status messages such as WM_DSPENGINE_COMPILEERROR and WM_DSPENGINE_RUNTIMEERROR sent by Engine Manager. The current definition of this structure is located in the DirectCore include file **enmgr.h**.

Each element in the structure description below includes a brief comment. See also the notes below, after the structure definition, and Signalogic Software Users Guide section Status Messages from DSP Engines, under section 7.4, Registering Callback Messages from DSP Engines.

typedef struct {

// type	// element	// comment
WORD	wErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

} ERRORMSGINFO;

typedef ERRORMSGINFO* PERRORMSGINFO;

Notes

1) The nErrorNum parameter can be interpreted as shown below:

Value	Meaning
>= 0	Error number; szErrorStatus contains error condition string
< 0	Indicates szEngineError contains pathname of error log file

2) Error log files are typically used in cases where the engine has found more than one error or warning.

3) The lParam value of the WM_DSPENGINE_xxxxERR messages points to an ERRORMSGINFO structure.

4) The wParam value of the WM_DSPENGINE_xxxxERR messages is the handle of the engine encountering the error.

See Also

DSRegisterEngineMsgWnd

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

Related Users Guide Topics

Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

ERRORMSGINFO Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_COMPILEERROR:

    int nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
```

```
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Compile Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

DSRegisterEngineMsgWnd

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

4.2 Constant Definitions

There are numerous constants defined in DSPower series software include files, such as enmgr.h, hwmgr.h, hwlib.h, and splib.h. The sections below describe these constant definitions.

 Copyright © Signalogic 1994-2013

5 Messages

Overview

This section provides a technical description of messages which can be sent and received when using DSPower series software. Messages have been divided into two categories:

- library messages
- DSP Engine messages

Library messages are generated by DSPower series libraries (DLLs, VBXs, OCXs, etc.) and sent to windows or controls owned by calling applications or by DSPower-Block Diagram software. These messages are essentially relays, or "reformats", of information contained in messages received from DSP Engines. Currently, messages are not sent from calling applications to DSPower series libraries; instead, various functions available in the library APIs cause the libraries to generate and send messages to DSP Engines. Note that to receive messages of any type from DSPower series libraries, calling applications must first "register" one or more window or control handles using the DSRegisterEngineMsgWnd function in Engine Manager.

By contrast, messages involving DSP Engines are transmitted directly, in both directions, between DSP Engines and other applications. Because different DSP Engines have different means of communicating with the outside world, the Windows library approach has been chosen to provide an intermediate layer and present a standardized programming interface. For example, Hypersignal-Macro series software sends information to a Windows virtual device driver (VxD), called Pipe.386, which arbitrates bi-directional shared memory command/data queues. Messages are formed as command and data information is extracted from these queues. As another example, MATLAB programs can send information using direct Windows API calls, via external library calls, or using a method based on the built-in MATLAB interactive control interface.

Thus, in DSPower series software, DSP Engine messages are normally processed automatically, without user-intervention. From the user's point of view--a C/C++ or MATLAB API in the case of DSPower series libraries, or a block diagram interface in the case of DSPower-Block Diagram software--these messages are not visible. However, DSP Engine messages have been fully documented to allow direct communication between user-defined applications and DSP Engines, if desired.

For general information on the rationale and benefits of the DSP Engine concept, see Signalogic Software System Guide section 1.1, Relationship of Signalogic Software Components.

Main Message Categories

Library Messages

DSP Engine Messages

5.1 Library Messages

The DSPower series library messages defined below can be received by calling applications which have implicitly or explicitly loaded Engine Manager (included with DirectCore software; see Signalogic Software Users Guide section 7.1, Engine Manager Access). There are some important items to note about DSPower series library messages:

1. Calling applications must first "register" one or more window or control handles using the DSRegisterEngineMsgWnd function in Engine Manager, before receiving these messages.
2. All library messages are standard Windows "WM_" messages (declared using the WM_USER constant), and not Windows notification messages (signified, for example, by "EN_", "CBN_", etc.). Notification messages are not used because all messages contain one or more pieces of data.
3. All library messages begin with "WM_DSPENGINE_".

Library Messages

WM_DSPENGINE_CODEGENERROR
WM_DSPENGINE_COMPILEERROR
WM_DSPENGINE_BUFNUM
WM_DSPENGINE_BUFSIZE
WM_DSPENGINE_BUFRDY
WM_DSPENGINE_ENGINESTATE
WM_DSPENGINE_FLGRDY
WM_DSPENGINE_FUNCTIONERROR
WM_DSPENGINE_LEVEL
WM_DSPENGINE_PROGSTATUS
WM_DSPENGINE_RUNTIMEERR
WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_CODEGENERROR

hEngine = (HENGINE) wParam // engine identifier
pErrorMsgInfo = (PERRORMSGINFO) lParam // pointer to error information

The WM_DSPENGINE_CODEGENERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered a format error or an in error in locating a native language source code file or executable file.

Parameter	Description
hEngine	handle of DSP Engine encountering code generation error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
WORD	wErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

If wErrorNum is positive, then it represents a DSP Engine native language source or executable file processing error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_CODEGENERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_CODEGENRROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Code Generation Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_COMPILEERROR

hEngine = (HENGINE) wParam // engine identifier
pErrorMsgInfo = (PERRORMSGINFO) lParam // pointer to error information

The WM_DSPENGINE_COMPILEERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered an error compiling a native language source code file.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
WORD	wErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

If wErrorNum is positive, then it represents a DSP Engine compilation error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the wErrorNum parameter is negative, then szStatusString is a pathname for an error log file. For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_COMPILEERROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Compile Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
```

```
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFNUM

```
hEngine = (HENGINE) wParam      // engine identifier
dwBufNum = (DWORD) lParam       // buffer number
```

The WM_DSPENGINE_BUFNUM engine update message is sent by Engine Manager after receiving a buffer number message from an engine executing a high-level data acquisition or generation function. The lParam parameter contains the current buffer number of the high-level generation function.

Parameter	Description
hEngine	handle of engine sending buffer number message
dwBufNum	current buffer number of high-level data acquisition or generation function

Returns

The calling application does not need to return a specific value.

Comments

The dwBufNum parameter indicates the current buffer number of the high-level data acquisition or generation function. Buffer numbers range from 0 to the maximum 32-bit word – 1, and increase monotonically at a constant rate dependent on the sampling frequency and buffer size being used in the high-level function. The current number of samples acquired (or generated) can be calculated by multiplying dwBufNum by the result obtained previously from a WM_DSPENGINE_BUFSIZE message.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_BUFNUM message being processed:

```
DWORD dwBufSize, dwBufNum;
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    break;

case WM_DSPENGINE_BUFSIZE:

    dwBufSize = lParam;  // store the buffer length being used
    break;

case WM_DSPENGINE_BUFNUM:

    dwCurSamplePos = dwBufSize*lParam;  // calculate current sample count

    char tmpstr[30], tmpstr2[30];
    ltoa(dwCurSamplePos, tmpstr, 10);  // convert to text
    ftoa(dwCurSamplePos/Fs, tmpstr2, 10);

    UpdateStatusBar(hSampleCounter, tmpstr, "Current Sample Count");
    UpdateStatusBar(hTimeCounter, tmpstr2, "Current Time (seconds)");
    break;
```

See Also

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DirectCore Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_BUFSIZE

hEngine = (HENGINE) wParam	// engine identifier
dwBufSize = (DWORD) lParam	// buffer size

The WM_DSPENGINE_BUFSIZE engine update message is sent by Engine Manager after receiving a buffer size message from a DSP Engine executing a high-level data acquisition, generation, or instrument function.

Parameter	Description
hEngine	handle of engine sending buffer size message
dwBufSize	buffer size being used during high-level data acquisition, generation or

instrument function

Returns

The calling application does not need to return a specific value.

Comments

The dwBufSize parameter indicates the buffer size being used during the high-level data acquisition, generation, or instrument function. Buffer sizes typically range from a few hundred samples up to 65535 samples.

Once the DSP Engine has calculated an appropriate buffer size, taking into account factors such as sampling rate, number of channels, and continuous signal generation requirements, the resulting value does not change for the duration of the high-level function. Note that in the case of continuous disk file acquisition or generation functions, it is typical for DSP Engines to calculate larger buffer sizes for high sampling rate values, in order to improve performance.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_BUFSIZE message being processed:

```
DWORD dwBufSize;
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    break;

case WM_DSPENGINE_BUFSIZE:

    dwBufSize = lParam;  // store the buffer length being used

    char tmpstr[30];
    ftoa(dwBufSize/Fs, tmpstr, 10);

    UpdateStatusBar(hTimeBase, tmpstr, "Framesize (in seconds)");
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DirectCore Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_BUFRDY

hEngine = (HENGINE) wParam // engine identifier
pDSPDataRdyMsgInfo = (PDSPDATARDYMSGINFO) lParam // pointer to buffer information

The WM_DSPENGINE_BUFRDY engine callback message is sent by Engine Manager after receiving a buffer ready message from a DSP Engine which has acquired, generated, or processed a multicore CPU or DSP/analog hardware data buffer.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pDSPDataRdyMsgInfo	pointer to PDSPDATARDYMSGINFO structure, which contains additional buffer information

Returns

The calling application does not need to return a specific value.

Comments

pDSPDataRdyMsgInfo points to a DSPDATARDYMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
WORD	wNum;	// buffer or flag number
DWORD	dwFlagValue;	// flag value
DWORD	dwFlagAddr;	// flag address
HGLOBAL	hCard;	// card handle

For more information, see section 5.1.3, DSPDATARDYMSGINFO Structure.

C/C++ Example

Below is an excerpt from the dscope.cpp source code example file (included with DirectCore software) showing a WM_DSPENGINE_BUFRDY message being received:

```
case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int*)GlobalLock(hBuf);  // lock memory to be used

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        if (nCurBuf == 0)  uStatus = DSReadMem(hCard, DS_RM_VECTOR_DATA_X, dwTimBufferBaseAddr, DS_RM_SIZE16, pBuf,
        wBuflen);

        else  uStatus = DSReadMem(hCard, DS_RM_VECTOR_DATA_Y, dwTimBufferBaseAddr, DS_RM_SIZE16, pBuf, wBuflen);
    }
    else { // linear data memory

        uStatus = DSReadMem(hCard, DS_RM_LINEAR_DATA_RT, dwTimBufferBaseAddr+nCurBuf*Buflen, DS_RM_SIZE16, pBuf,
        wBuflen);
    }

    GlobalUnlock(hBuf);  // let memory float
```

```
if (!uStatus) MessageBox(NULL, "DSReadMem problem", "Test Prog", MB_OK);

// switch buffers and write new buffer number

nCurBuf ^= 1;
DSSetProperty(hCard, DSPROP_HOSTBUFNUM, nCurBuf);

// can we have another please, Ma'am..
DSWaitForBuffer(hCard, nCurBuf, NULL, DS_WFB_POLLED);
break;
```

A significant amount of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the entire dscope.cpp file be examined before implementing code similar to the above example. For example, uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

In the above example, hBuf is assumed to have been created via a GlobalAlloc call, using a sizeof parameter equal to 2∗wBuflen. Memory pointers passed to DSReadMem and DSWriteMem calls *must not point to memory segments less than the product of the dwCount and DS_RM_SIZExx parameters* used in the corresponding DSReadMem or DSWriteMem calls. This product is in bytes.

See Also

WM_DSPENGINE_FLGRDY

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

WM_DSPENGINE_ENGINESTATE

WM_DSPENGINE_ENGINESTATE

hEngine = (HENGINE) wParam // engine identifier
wEngineState = LOWORD(lParam) // current engine state

The WM_DSPENGINE_ENGINESTATE engine status message is sent by Engine Manager after receiving a status message from a DSP Engine indicating its current state, or when Engine Manager suspends or resumes an engine.

Parameter	Description
hEngine	Handle of engine for which engine state message applies
wEngineState	Current state of engine, as defined by one of the values below:

Value	Meaning
DS_ES_RUNNING	engine is in running (busy) state
DS_ES_IDLE	engine is in idle state

DS_ES_STOPPENDING	engine is running, but a stop (shutdown) command is pending
DS_ES_STOPPED	engine is stopped (shutdown)

Returns

The calling application does not need to return a specific value.

Comments

The wEngineState parameter reflects the current state of the DSP Engine. Note that receiving a WM_DSPENGINE_ENGINESTATE message does not necessarily mean there has been a state change. To detect changes, the calling application should update a local variable as part of its WM_DSPENGINE_ENGINESTATE message processing.

C/C++ Example

An example of processing a WM_DSPENGINE_ENGINESTATE message is shown below:

```
WORD static wPrevEngineState = DS_ES_IDLE;
WORD wEngineState;

case WM_DSPENGINE_ENGINESTATE:

    wEngineState = LOWORD(lParam);

    if (wEngineState == DS_ES_IDLE && wEngineState != wPrevEngineState) {

        UpdateStatusBar(hState, "Idle");  // engine state returned to idle
    }
    wPrevEngineState = wEngineState;  // update local copy of engine state
    break;
```

See Also

WM_DSPENGINE_PROGSTATUS

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_FLGRDY

WM_DSPENGINE_FLGRDY

hEngine = (HENGINE) wParam	// engine identifier
pDSPDataRdyInfo = (PDSPDATARDYMSGINFO) lParam	// pointer to flag information

The WM_DSPENGINE_FLGRDY engine callback message is sent by Engine Manager after receiving a buffer ready message from a DSP Engine which has acquired, generated, or processed a multicore CPU or DSP/analog hardware data buffer.

Parameter	Description
hEngine	Handle of DSP Engine encountering compiler error
pDSPDataRdyInfo	Pointer to PDSPDATARDYMSGINFO structure, which contains additional flag information

Returns

The calling application does not need to return a specific value.

Comments

pDSPDataRdyMsgInfo points to a DSPDATARDYMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
WORD	wNum;	// buffer or flag number
DWORD	dwFlagValue;	// flag value
DWORD	dwFlagAddr;	// flag address
HGLOBAL	hCard;	// card handle

For more information, see section 5.1.3, DSPDATARDYMSGINFO Structure.

C/C++ Example

Below is an example showing a WM_DSPENGINE_FLGRDY message being received:

```
case WM_DSPENGINE_FLGRDY:  // semaphore message sent by DSP engine

    DWORD dwCount = 4096;  // process 4096 floating-point values each frame

    HGLOBAL hBuf = GlobalAlloc(GMEM_INIT | GMEM_MOVEABLE, dwCount*DS_RM_SIZE32);

    (float*) pBuf = (float*)GlobalLock(hBuf);  // lock memory before use

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        uStatus = DSReadMem(hCard, DS_RM_VECTOR_DATA_X, dwResultDataAddr, DS_RM_SIZE32, pBuf, dwCount);
    }
    else { // linear data memory

        uStatus = DSReadMem(hCard, DS_RM_LINEAR_DATA_RT, dwResultDataAddr, DS_RM_SIZE32, pBuf, dwCount);
    }

    GlobalUnlock(hBuf);  // let memory go

    if (!uStatus) {

        // error occurred in DSReadMem call
        MessageBox(NULL, "DSReadMem problem", "Example", MB_OK);
    }
    else {

        // wait for semaphore again...
        DSWaitForFlag(hCard, 0, NULL, DS_WFB_POLLED);
    }
    break;
```

A great deal of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the dscope.cpp example source code file (included with DirectCore

software) be examined as at least one good illustration before implementing code similar to the above example. For example, uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

In the above example, note that hBuf is created via a GlobalAlloc call, using a sizeof parameter equal to the total element count*memory width per element. Memory pointers passed to DSReadMem and DSWriteMem calls *must not point to memory segments less than the product of the dwCount and DS_RM_SIZExx parameters* used in the corresponding DSReadMem or DSWriteMem calls. The resulting product is in bytes.

See Also

WM_DSPENGINE_BUFRDY

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_FUNCTIONERROR

hEngine = (HENGINE) wParam // engine identifier
pErrorMsgInfo = (PERRORMSGINFO) lParam // pointer to error information

The WM_DSPENGINE_FUNCTIONERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine for which a function, display, or instrument has encountered an input parameters or other setup error. This error message is sent before the function starts.

Parameter	Description
hEngine	handle of DSP Engine encountering function error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
WORD	wErrorNum;	// error number or type
Char	szErrorStatus[256];	// error/status string or error log filename

If wErrorNum is positive, then it represents a DSP Engine function initialization error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only

produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_FUNCTIONERROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Function Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_LEVEL

WM_DSPENGINE_LEVEL

hEngine = (HENGINE) wParam // engine identifier
dwLevel = (DWORD) lParam // maximum amplitude level

The WM_DSPENGINE_LEVEL engine update message is sent by Engine Manager when a DSP Engine amplitude level message is received during a high-level data acquisition, generation, or instrument function. The lParam parameter contains the maximum absolute amplitude found in each buffer of acquired or generated data.

Parameter	Description
hEngine	handle of engine sending level message
dwLevel	maximum amplitude level found in each buffer during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The dwLevel parameter indicates the maximum absolute value found in each buffer of data processed during the high-level data acquisition, generation, or instrument function. Values range from 0 to the maximum full-scale value of the A/D or D/A converter(s) being used in the high-level function.

Note that values are always positive.

C/C++ Example

Below is an example of processing a WM_DSPENGINE_LEVEL message:

```
case WM_DSPENGINE_LEVEL:

    UpdateMeter(hMeter, lParam);  // lParam has maximum amplitude
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_SAMPFREQ

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DirectCore Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_PROGSTATUS

WM_DSPENGINE_PROGSTATUS

```
hEngine = (HENGINE) wParam          // engine identifier
wProgramStatus = LOWORD(lParam)     // program status
wFunctionID = HIWORD(lParam)        // function ID
```

The WM_DSPENGINE_PROGSTATUS program status message is sent by Engine Manager after receiving a program status message from a DSP Engine containing status information regarding an executing program.

Parameter	Description
hEngine	Handle of engine for which program status information applies
wProgramStatus	Program status, as defined by one of the values below:

Value	Meaning
DS_PS_FUNCDONE	engine function is complete
DS_PS_PROGDONE	program is complete

wFunctionID	Identifier of function, if wProgramStatus above indicates a function has completed

Returns

The calling application does not need to return a specific value.

Comments

Depending on its flow of execution, a DSP Engine program may cause several DSP Engine functions to execute. Processing WM_DSPENGINE_PROGSTATUS functions provides a method to dynamically monitor the status of DSP Engine programs, regardless of the execution variations which are data-dependent or user-dependent.

Note that function ID values must be encoded into the source code generated and submitted to the DSP Engine to be executed as a native language program. This is done using the SetFunctionID DSP Engine function, which sets the current function ID for use in DSP Engine communication with the outside world.

Note that WM_DSPENGINE_PROGSTATUS program status callback messages are sent before any library functions, such as DSSendEngineCommand or DSExecEngineFunction,. which might be waiting for "program done" notification, are released. This situation can occur if these functions were executed by the calling application in asynchronous mode (uMode parameter == DS_EEF_ASYNC).

C/C++ Example

An example of processing a WM_DSPENGINE_PROGSTATUS message is shown below:

```
case WM_DSPENGINE_PROGSTATUS:

    if (LOWORD(lParam) == DS_PS_PROGDONE) {

        UpdateStatusBar(hProgStatus, "Engine Program Complete");
    }
    break;
```

See Also

DSRegisterEngineMsgWnd

DSExecEngineFunction

WM_DSPENGINE_ENGINESTATE

Related Users Guide Topics

Program Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_RUNTIMEERROR

WM_DSPENGINE_RUNTIMEERROR

hEngine = (HENGINE) wParam // engine identifier

pErrorMsgInfo = (PERRORMSGINFO) lParam // pointer to error information

The WM_DSPENGINE_RUNTIMEERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered an error while executing a program.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

pErrorMsgInfo points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
WORD	wErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

If wErrorNum is positive, then it represents a DSP Engine run-time error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the wErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative wErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_RUNTIMEERROR message being received:

```
char szErrorStatus[256], char tmpstr[30];

case WM_DSPENGINE_RUNTIMEERROR:

    _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);
    _fstrcat(szErrorStatus, "; Error Number = ");
    itoa((PERRORMSGINFO)lParam->nErrorNum, tmpstr, 10);
    _fstrcat(szErrorStatus, tmpstr);

    MessageBox(NULL, szErrorStatus, "Run-Time Error Message", MB_OK);
```

```
    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_SAMPFREQ

```
hEngine = (HENGINE) wParam      // engine identifier
SampFreq = (float) lParam       // sampling frequency
```

The WM_DSPENGINE_SAMPFREQ engine update message is sent by Engine Manager when a DSP Engine "actual sampling frequency" message is received during a high-level data acquisition, generation, or instrument function. The lParam parameter contains the actual sampling frequency being used during the function.

Parameter	Description
hEngine	handle of DSP Engine sending sampling frequency message
SampFreq	sampling frequency being used during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The SampFreq parameter indicates the sampling frequency, in Hz, being used during the high-level data acquisition, generation, or instrument function. Sampling frequency values typically range from a few Hz up to 10 MHz or higher.

Although all high-level DSP Engine functions include a desired sampling rate parameter, the actual rate used may differ. This occurs because many types of multicore CPU or DSP/analog hardware offer only a limited number of discrete sampling rate values. DSP Engine high-level functions then choose the physical value closest to the desired value, and report the choice using the WM_DSPENGINE_SAMPFREQ message.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_SAMPFREQ message being processed:

```
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    char tmpstr[30];
    gcvt(Fs, 10, tmpstr);

    UpdateStatusBar(hSampFreq, tmpstr, "Sampling Frequency (in Hz)");
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DirectCore Software

section 7.3 Executing Functions in DSP Engines

 Copyright © Signalogic 1994-2013

5.2 DSP Engine Messages

This section documents messages sent by DSP Engines. Normally, in DSPower series software, these messages are processed automatically, without user-intervention. For example, in response to a DSEngineOpen call, the DirectCore library receives and sends engine messages to Hypersignal-Macro series software in order to initialize the software in engine mode. From the user's point of view, in this case a C/C++ or MATLAB API, these messages are not visible--only the library call, in this case DSEngineOpen, is visible. Another example is the DSPower-Block Diagram package, which either processes DSP Engine messages directly or uses an intermediate layer such as the DirectCore library. Again, from the user's point of view, engine messages are not visible--instead, an easy-to-use, graphical block diagram user-interface is visible.

For more information on the rationale and benefits of the DSP Engine concept, see Signalogic Software System Guide section 1.1, Relationship of Signalogic Software Components.

The actual numeric constants defined for DSP Engine messages are contained in the include files enmgr.h and engdef.h, which are supplied with DirectCore software, and which may be more current than this documentation.

DSP Engine Messages

DSPENG_ABORT

DSPENG_BUFRDY

DSPENG_CALCSAMPFREQ

DSPENG_CALCSAMPFREQRESULT

DSPENG_CANCELWAIT

DSPENG_CLIPRECT

DSPENG_CODEGENERROR

DSPENG_COMPILEERROR

DSPENG_DISABLECARD

DSPENG_DISPDIRFILESTEXT

DSPENG_DISPSTATUSTEXT

DSPENG_DISPWVFRMSUMTEXT

DSPENG_ERASERECT

DSPENG_ESC

DSPENG_EXIT

DSPENG_FLGRDY

DSPENG_FRAMERECT

DSPENG_FUNCTIONACTIVE

DSPENG_FUNCTIONDONE

DSPENG_FUNCTIONERROR

DSPENG_FUNCTIONPAUSE

DSPENG_GETCARDCLASS

DSPENG_GETCARDCLASSRESULT

DSPENG_GETPROPERTY

DSPENG_GETPROPERTYMEMRESULT

DSPENG_GETMEM

DSPENG_GETMEMRESULT

DSPENG_GETMEMSIZE
DSPENG_GETMEMSIZERESULT
DSPENG_GETPARMS
DSPENG_HOLDCARD
DSPENG_INITCARD
DSPENG_INITCARDSTATUS
DSPENG_LINEREL
DSPENG_LINETO
DSPENG_LOADFILE
DSPENG_LOADFILESTATUS
DSPENG_MOVEREL
DSPENG_MOVETO
DSPENG_PAINTRECT
DSPENG_PUTPROPERTYMEM
DSPENG_PUTMEM
DSPENG_PUTMEMRESULT
DSPENG_RDCARDREG
DSPENG_RDCARDREGRESULT
DSPENG_REDRAW
DSPENG_REFRESH
DSPENG_REPORTBUFNUM
DSPENG_REPORTBUFSIZE
DSPENG_REPORTDISPID
DSPENG_REPORTFUNCID
DSPENG_REPORTLEVEL
DSPENG_REPORTSAMPFREQ
DSPENG_RESETCARD
DSPENG_RUNCARD
DSPENG_RUNEXTPROGASYNC
DSPENG_RUNEXTPROGSYNC
DSPENG_RUNTIMEERROR
DSPENG_SETAMP
DSPENG_SETBACKCOLOR
DSPENG_SETBITMAP
DSPENG_SETCARDBASEADDR
DSPENG_SETBUFNUMID
DSPENG_SETCOLOR
DSPENG_SETCURTRACE

DSPENG_SETDISPID
DSPENG_SETDISPLAY
DSPENG_SETFUNCID
DSPENG_SETGCUR
DSPENG_SETLEVELID
DSPENG_SETOFS
DSPENG_SETPENCOLOR
DSPENG_SETPENMODE
DSPENG_SETRES
DSPENG_SETPIXEL
DSPENG_SETSTATISTICSID
DSPENG_SETTRACE
DSPENG_SETUNITS
DSPENG_SETUP
DSPENG_SETVOLTAGERANGE
DSPENG_SHUTDOWN
DSPENG_WAITFORBUF
DSPENG_WAITFORFLG
DSPENG_WAVEFORMEDITED
DSPENG_WRCARDREG
WM_DSPENGINE_CMDDATA

DSPENG_ABORT

// command instructing engine to abort and unload from system, without saving currently open files or configuration

DSPENG_BUFRDY

// engine is indicating that buffer previously assigned by DSPENG_WAITFORBUF command is ready

DSPENG_CALCSAMPFREQ

// command instructing engine to calculate actual sampling frequency for specified multicore CPU or DSP/analog hardware

DSPENG_CALCSAMPFREQRESULT

// engine is sending result of DSPENG_CALCSAMPFREQ command

DSPENG_CANCELWAIT

// command instructing engine to cancel pending wait requests

DSPENG_CLIPRECT

// engine is sending command to set clipping rectangle

DSPENG_CODEGENERROR

// engine encountered code generation error (WORD error number)

DSPENG_COMPILEERROR

// engine encountered compile error (WORD error number)

DSPENG_DISABLECARD

// command instructing engine to disable specified multicore CPU or DSP/analog hardware

DSPENG_DISPDIRFILESTEXT

// engine is sending file directory status for display

DSPENG_DISPSTATUSTEXT

// engine is sending status text for display

DSPENG_DISPWVFRMSUMTEXT

// engine is sending waveform summary text for display

DSPENG_ERASERECT

// engine is sending command to erase rectangle

DSPENG_ESC

// command instructing engine to escape back up one level

DSPENG_EXIT

// command instructing engine to save open data and configuration files and unload from system

DSPENG_FLGRDY

// engine is indicating that flag previously assigned by DSPENG_WAITFORFLG command is ready

DSPENG_FRAMERECT

// engine is sending command to draw rectangle frame

DSPENG_FUNCTIONACTIVE

// engine function or program active

DSPENG_FUNCTIONDONE

// engine function or program complete

DSPENG_FUNCTIONERROR

// engine function or program terminated with error

DSPENG_FUNCTIONPAUSE

// engine function or program has paused

DSPENG_GETCARDCLASS

// command instructing engine to return driver classification for specified multicore CPU or DSP/analog hardware

DSPENG_GETCARDCLASSRESULT

// engine is sending result of DSPENG_GETCARDCLASS command

DSPENG_GETPROPERTYMEM

// command instructing engine to read RTAF property

DSPENG_GETPROPERTYMEMRESULT

// engine is sending result of DSPENG_GETPROPERTYMEM command

DSPENG_GETMEM

// command instructing engine to read data from DSP memory; for specified multicore CPU or DSP/analog hardware

DSPENG_GETMEMRESULT

// engine is reporting completion status of previous DSPENG_GETMEM command

DSPENG_GETMEMSIZE

// command instructing engine to calculate maximum data memory size for specified multicore CPU or DSP/analog hardware

DSPENG_GETMEMSIZERESULT

// engine is sending result of DSPENG_GETMEMSIZE size command

DSPENG_GETPARMS

// engine can accept parameters

DSPENG_HOLDCARD

// command instructing engine to place specified multicore CPU or DSP/analog hardware in hold mode

DSPENG_INITCARD

// command instructing engine to initialize specified multicore CPU or DSP/analog hardware

DSPENG_INITCARDSTATUS

// engine is reporting result of DSPENG_INITCARD command (multicore CPU or DSP/analog hardware initialization status)

DSPENG_LINEREL

// engine is sending command to draw line relative

DSPENG_LINETO

// engine is sending command to draw line absolute

DSPENG_LOADFILE

// command instructing engine to download program file for specified multicore CPU or DSP/analog hardware

DSPENG_LOADFILESTATUS

// engine is reporting completion status of DSPENG_LOADFILE command

DSPENG_MOVEREL

// engine is sending command to move graphics position relative

DSPENG_MOVETO

// engine is sending command to move graphics position absolute

DSPENG_PAINTRECT

// engine is sending command to paint rectangle

DSPENG_PUTPROPERTYMEM

// command instructing engine to write RTAF property

DSPENG_PUTMEM

// command instructing engine to write data to DSP memory for specified multicore CPU or DSP/analog hardware

DSPENG_PUTMEMRESULT

// engine is reporting completion status of previous DSPENG_PUTMEM command

DSPENG_RDCARDREG

// command instructing engine to read card register for specified multicore CPU or DSP/analog hardware

DSPENG_RDCARDREGRESULT

// engine is sending result of DSPENG_RDCARDREG command

DSPENG_REDRAW

// command instructing engine to return to initial frame parameters

DSPENG_REFRESH

// command instructing engine to refresh display

DSPENG_REPORTBUFNUM

// engine is reporting current multicore CPU or DSP code buffer count for high-level function

DSPENG_REPORTBUFSIZE

// engine is reporting multicore CPU or DSP code buffer size being used in high-level function

DSPENG_REPORTDISPID

// engine is reporting new active display ID

DSPENG_REPORTFUNCID

// engine is reporting new active function ID

DSPENG_REPORTLEVEL

// engine is reporting current maximum absolute amplitude level for data buffer in high-level function

DSPENG_REPORTSAMPFREQ

// engine is reporting multicore CPU or DSP/analog hardware sampling frequency being used in high-level function

DSPENG_RESETCARD

// command instructing engine to reset specified multicore CPU or DSP/analog hardware

DSPENG_RUNCARD

// command instructing engine to run specified multicore CPU or DSP/analog hardware

DSPENG_RUNEXTPROGASYNC

// run external program asynchronously with engine

DSPENG_RUNEXTPROGSYNC

// run an external program synchronously with engine

DSPENG_RUNTIMEERROR

// engine encountered run-time error (WORD error number)

DSPENG_SETAMP

// engine is sending command to set amplitude "hotspot" area

DSPENG_SETBACKCOLOR

// engine is sending command to set background color

DSPENG_SETBITMAP

// engine is sending command to set current bitmap

DSPENG_SETCARDBASEADDR

// command instructing engine to set I/O and memory base addresses for specified multicore CPU or DSP/analog hardware

DSPENG_SETBUFNUMID

// command instructing engine to set buffer number message ID

DSPENG_SETCOLOR

// command instructing engine to set current drawing color

DSPENG_SETCURTRACE

// engine requesting to set current display trace in multitrace display

DSPENG_SETDISPID

// command instructing engine to set current display ID

DSPENG_SETDISPLAY

// engine is sending command to set current display page

DSPENG_SETFUNCID

// command instructing engine to set current function ID

DSPENG_SETGCUR

// command instructing engine to set current graphics cursor

DSPENG_SETLEVELID

// command instructing engine to set amplitude level message ID

DSPENG_SETOFS

// command instructing engine to set current graphics offset

DSPENG_SETPENCOLOR

// engine is sending command to set pen color

DSPENG_SETPENMODE

// engine is sending command to set pen mode

DSPENG_SETRES

// command instructing engine to set current graphics resolution

DSPENG_SETPIXEL

// engine is sending command to set pixel

DSPENG_SETSTATISTICSID

// command instructing engine to set statistics message ID

DSPENG_SETTRACE

// engine is sending command to set current trace "hotspot" area

DSPENG_SETUNITS

// engine is sending command to set units "hotspot" area

DSPENG_SETUP

// engine sends this message after initializing and entering command mode

DSPENG_SETVOLTAGERANGE

// command instructing engine to set voltage ranges for specified multicore CPU or DSP/analog hardware

DSPENG_SHUTDOWN

// engine sends this command as last action before shutdown

DSPENG_WAITFORBUF

// command instructing engine to assign wait-for-buffer request

DSPENG_WAITFORFLG

// command instructing engine to assign wait-for-flag request

DSPENG_WAVEFORMEDITED

// engine reports waveform has been edited

DSPENG_WRCARDREG

// command instructing engine to write card register for specified multicore CPU or DSP/analog hardware

WM_DSPENGINE_CMDDATA

WM_DSPENGINE_CMDDATA

wMsgType = wParam // message type
pEngineSharedInfo = (PENGINESHAREDINFO) lParam // pointer to shared memory

The WM_DSPENGINE_CMDDATA message is sent by DSP Engines to external applications. Depending on the engine, it can be sent using a VxD or using direct Windows API SendMessage, PostMessage, or PostAppMessage calls.

Parameter	Description
wMsgType	Message type, as defined below:

Value	Meaning
DS_CD_VXDMSG	message sent via VxD
DS_CD_APIMSG	message sent from standard API call

pEngineSharedInfo	Pointer to shared memory, which can be accessed receiving application using ENGINESHAREDINFO structure definition in engdef.h

Returns

The calling application does not need to return a specific value.

Comments

The WM_DSPENGINE_CMDDATA message contains a pointer to a shared memory area. Normally, this shared area contains bi-directional queues in which the external application and DSP Engine can exchange command and data information at maximum transfer rates. The bi-directional queues allow exchange of "command packets", which contain variable length data elements depending on the nature of the command. It is these commands which form many of the possible DSP Engine messages.

C/C++ Example

An example of processing a WM_DSPENGINE_CMDDATA message is shown below:

```
case WM_DSPENGINE_CMDDATA:

    pSharedEngineInfo = (PENGINESHAREDINFO)lParam;

    while (!IsQueueEmpty()) {

        Cmd = GetQueueByte();
```

```
        switch (Cmd) {

            // process commands
        }
    }
    break;
```

See Also

ENGINESHAREDINFO

DSExecEngineFunction

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

 Copyright © Signalogic 1994-2013

6 Error and Status Conditions

Overview

The Error and Status Conditions section provides listings and technical descriptions of errors and status conditions, text displays, and reports which can occur in DSPower series and Hypersignal-Macro series software. Error and status conditions are further subdivided into the following categories:

- DSPower-Block Diagram
- DLL Functions
- DSP Engines

While DSPower-Block Diagram error and status conditions may cause display of an error number for documentation purposes, these conditions do not currently produce numeric constants which can be utilized by C/C++ or DSP Engine native language programs.

By contrast, DLL function error and status conditions are defined as numeric constants, which are returned by the functions themselves or by DSxxxGetErrorStatus routines located in various DLLs. They can also be accompanied by error and status text displays and reports which appear in the Engine Manager error/status listbox, if enabled (see Signalogic Software Users Guide section 7.5.3, Error/Status Window in Engine Manager).

NOTE: In some extreme cases, a DSPower series DLL may also independently display its own message box, which indicates a serious system error, and which should be carefully noted and reported to Signalogic if it occurs.

DSP Engine error and status conditions and text displays are generated and returned by programs such as Hypersignal-Macro series software or MATLAB, even though they may appear in status boxes or listboxes maintained by DSPower-Block Diagram software or a DSPower series DLL. For example, Engine Manager, which is part of DirectCore software, offers an error/status listbox which can be displayed as a debug tool, and which shows error and status conditions and text displays generated by DSP Engines as well as itself and other DSPower series DLLs. For more information, see Signalogic Software Users Guide section 7.5.3, Error/Status Window in Engine Manager.

Error and Status Condition Categories

DSPower-Block Diagram Error and Status Conditions

DLL Error and Status Conditions

DSP Engine Error Conditions

See Also

Error and Status Condition Numeric List and Cross Reference

Below is a numeric listing of all error and status conditions, cross-referenced by page number.

Error and Status Condition Numeric List and Cross Reference

In the table below, there are two subtleties worth pointing out. First, positive error values indicate errors that could reasonably be expected to happen due to operator error, and should be recoverable without difficulty. Negative error values indicate a more serious level of error, such as necessary program files not installed correctly, Windows resource problems, incorrect calling application usage of DLL functions, or internal DSPower series DLL errors.

Second, error values are grouped according to function. Error values -1 through -9 are general DLL errors (located in dllerr.h). Error values -10 through -49 can be generated by Engine Manager (enmgr.dll, located in enmgr.h), values -50 through -99 by Hardware Library (hwlib.dll, located in hwlib.h), and values -100 through -149 by Hardware Manager (hwmgr.dll, located in hwmgr.h).

Number	**Error Definition**	**Page**
11	DSDSPFILENOTFOUND	207
10	DSHARDWARENOTRESPONDING	207
2	DSCOULDNOTOPENENGINE	203
1	DSINVALIDENGINEPATH	204
-1	DSCOULDNOTCREATEBKGNDWINDOW	201
-2	DSCOULDNOTSTARTBKGNDTASK	201
-3	DSNOBKGNDWINDOW	202
-4	DSCOULDNOTINITIALIZEDLL	202
-10	DSENGINETIMEOUT	204
-11	DSCOULDNOTOPENPIPE	204
-12	DSCOULDNOTSTARTENGINE	204
-13	DSNOENGINESOPEN	204
-14	DSNOENGINEFOUNDINLIST	204
-15	DSENGINEHANDLEWASNULL	204
-16	DSENGINEMEMORYDISCARDED	204
-50	DSINVALIDCARDHANDLE	207
-51	DSCOULDNOTOPENTEMPLATEFILE	207
-52	DSCOULDNOTCREATESOURCEFILE	207
-53	DSCREATINGENGINEPROGRAMERROR	207
-54	DSUNABLETOALLOCATEMEMORY	207
-55	DSUNABLETOREADTEMPLATEFILE	207
-56	DSERRORINDATAPARAM	207
-57	DSCARDDOESNOTSUPPORTCALL	208
-58	DSINVALIDWORDLENGTH	208
-59	DSINVALIDMEMORYTYPE	208
-60	DSINVALIDCARDDESIGNATOR	208
-61	DSCARDNOTINHWSETUPFILE	208
-62	DSALLCARDHANDLESALLOCATED	208
-63	DSINVALIDCORENUMBER	208
-100	DSUNABLETOREADHWSETUPFILE	205
-101	DSCOULDNOTLOCKHWLIST	205

-102	DSCARDDESIGNATORNOTLISTED	206
-103	DSCOULDNOTFINDHWSETUPFILE	206
-104	DSCOULDNOTINCREASEHWLISTMEM	206
-105	DSHWSETUPFILENOTYETREAD	206

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DirectCore Software

 Copyrght © Signalogic 1994-2013

6.2 DLL Error and Status Conditions

This section defines the DLL error and status numerical constants which can be returned by functions in DSPower series DLLs.

In the DSPower series DLL error-handling scheme, a relatively small number of error and status constants are returned directly by the function called. However, most functions either return NULL, or zero, to indicate failure, and TRUE, or non-zero, to indicate success. Often times, the non-zero value returned on success is the actual return value of the function itself. Obvious examples include the DSEngineOpen and DSAssignCard calls in DirectCore software, which return engine and card handles, respectively.

Each DLL has a "base level" error/status routine which can be used to investigate error conditions. Currently, these calls are as follows:

Error/Status Call	**DLL**
DSGetEngMgrErrorStatus	Engine Manager (enmgr.dll)
DSGetHWMgrErrorStatus	Hardware Manager (hwmgr.dll)
DSGetHWLibErrorStatus	Hardware Library (hwlib.dll)
DSGetSPLibErrorStatus	Signal Processing Library (splib.dll)

In some cases, there may be additional "higher-level" error/status routines available. For example, Engine Manager also offers the DSGetEngineErrorStatus call to investigate problems associated with a specific engine; this call requires a valid engine handle as an input parameter.

Note also that error and status numerical constants are not the whole story. Engine Manager (enmgr.dll) maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of both status and error conditions resulting from most Engine Manager and Hardware Manager calls. For information about this status window, see Signalogic Software Users Guide section 7.5.3, Error/Status Window in Engine Manager. For complete information on error handling in Engine Manager, see Signalogic Software Users Guide section 7.5, Error Handling in Engine Manager.

NOTE: In some extreme cases, a DSPower series DLL may also independently display its own message box, which indicates a serious system error, and which should be carefully noted and reported to Signalogic if it occurs.

DLL Error and Status Message Categories

 Copyright © Signalogic 1994-2013

6.2.1 General DLL Error and Status Conditions

Below is a source code example of using the Engine Manager "base level" error/status routine. The shows further investigation after a DirectCore software DSEngineOpen call failure:

```
if (!DSEngineOpen(szEnginePath, szWaveformPath)) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSEngineOpen would be NULL in the case of failure, since a non-NULL value would be a valid engine handle. Thus, to find out more information, the DSGetEngMgrErrorStatus call is used.

Below is a listing of ***general DLL*** error values which can be returned by DSGetEngMgrErrorStatus, along with a brief description of their meaning. These constants can be found in the dllerr.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions.

Note that it is possible in some circumstances for the error values listed below to also be returned by the DSGetHWMgrErrorStatus, DSGetHWLibErrorStatus, and DSGetSPLibErrorStatus routines, besides DSGetEngMgrErrorStatus. Note also that in C/C++ programs, it is safest to treat these error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

General DLL Error Values

DSCOULDNOTCREATEBKGNDWINDOW

DSCOULDNOTSTARTBKGNDTASK

DSNOBKGNDWINDOW

DSCOULDNOTINITIALIZEDLL

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DirectCore Software

DSCOULDNOTCREATEBKGNDWINDOW

The background (hidden) window needed by enmgr.dll could not be created.

DSCOULDNOTSTARTBKGNDTASK

The background (hidden) task needed by enmgr.dll could not be started.

Copyrght © Signalogic 1994-2013

DSNOBKGNDWINDOW

The command failed because there was no background window.

DSCOULDNOTINITIALIZEDLL

Enmgr.dll itself could not be initialized.

 Copyright © Signalogic 1994-2013

6.2.2 Engine Manager Error and Status Conditions

Below is a source code example of using the Engine Manager "base level" error/status routine. The shows further investigation after a DSEngineOpen failure:

```
if (!DSEngineOpen(szEnginePath, szWaveformPath)) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSEngineOpen would be NULL in the case of failure, since a non-NULL value would be a valid engine handle. Thus, to find out more information, the DSGetEngMgrErrorStatus call is used.

Below is a listing of error values returned by DSGetEngMgrErrorStatus, along with a brief description of their meaning. These constants can be found in the enmgr.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions. Note that in C/C++ programs, it is safest to treat these error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

Engine Manager Error Values

DSCOULDNOTOPENENGINE

DSINVALIDENGINEPATH

DSENGINETIMEOUT

DSCOULDNOTOPENPIPE

DSCOULDNOTSTARTENGINE

DSNOENGINESOPEN

DSNOENGINEFOUNDINLIST

DSENGINEHANDLEWASNULL

DSENGINEMEMORYDISCARDED

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DirectCore Software

DSCOULDNOTOPENENGINE

Probably indicates DSP Engine executable file could not be located.

DSINVALIDENGINEPATH

An invalid engine path was specified in a DSEngineOpen call.

DSENGINETIMEOUT

Time-out limits were exceeded in waiting for the DSP Engine to successfully open and initialize, or respond to a command or request.

DSCOULDNOTOPENPIPE

The Pipe.386 VxD could not be opened for a DOS DSP Engine. Make sure that pipe.386 is listed in the "386Enh" section of the system.ini file on the x:\windows subdirectory.

DSCOULDNOTSTARTENGINE

The DSP Engine could not be initialized correctly.

DSNOENGINESOPEN

A call was made requiring an engine handle but no engines were open.

DSNOENGINEFOUNDINLIST

A call was made using an engine handle which didn't specify an engine currently being maintained by enmgr.dll.

DSENGINEHANDLEWASNULL

A call was made using a NULL engine handle.

DSENGINEMEMORYDISCARDED

A call was made using an engine handle which pointed to memory already discarded.

 Copyright © Signalogic 1994-2013

6.2.3 Hardware Manager Error and Status Conditions

Below is a source code example of using the Hardware Manager "base level" error/status routine. The shows further investigation after a DSSetHWMgrCurVal failure:

```
if (!DSSetHWMgrCurVal(szCardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSSetHWMgrCurVal would be NULL in the case of failure. Thus, to find out more information, the DSGetHWMgrErrorStatus call is used.

Below is a listing of error values returned by DSGetHWMgrErrorStatus, along with a brief description of their meaning. These constants can be found in the hwmgr.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions. Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

Hardware Manager Error Values

DSUNABLETOREADHWSETUPFILE

DSCOULDNOTLOCKHWLIST

DSCARDDESIGNATORNOTLISTED

DSCOULDNOTFINDHWSETUPFILE

DSCOULDNOTINCREASEHWLISTMEM

DSHWSETUPFILENOTYETREAD

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DirectCore Software

DSUNABLETOREADHWSETUPFILE

Unable to read the Hardware Setup File.

DSCOULDNOTLOCKHWLIST

Unable to lock the global memory hardware list maintained inside hwmgr.dll

 Copyrght © Signalogic 1994-2013

DSCARDDESIGNATORNOTLISTED

A card designator input to a hwmgr.dll function was not listed in the Hardware Setup File.

DSCOULDNOTFINDHWSETUPFILE

The Hardware Setup File could not be located.

DSCOULDNOTINCREASEHWLISTMEM

The global memory hardware list maintained inside hwmgr.dll could not be increased in size (to accommodate additional Hardware Setup File entries).

DSHWSETUPFILENOTYETREAD

A function in Hardware Manager was called which needed access to Hardware Setup File data, but the DSReadHWSetupFile function had not yet been called.

 Copyright © Signalogic 1994-2013

6.2.4 Hardware Library Error and Status Conditions

The DirectCore error handling scheme is straightforward. Most library calls return a non-zero value on success, and a zero (NULL) on failure. When calls into hwlib.dll fail, the DSGetHWLibErrorStatus and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure. These calls return error and status conditions associated with multicore CPU or DSP/analog hardware and DSP/math functions.

Generally, at any given time both of these calls will return the same value. However, there is an important difference: the latter retrieves error conditions associated with a particular engine, while the former returns an overall DLL value. Thus, if more than one engine is open at one time, it is possible that the DSGetHWLibErrorStatus value could be overwritten as a result of consecutive unrelated engine calls.

For error and status conditions associated with engine problems, the DSGetEngMgrErrorStatus and DSGetEngineErrorStatus routines (located in enmgr.dll) should be used. For error and status conditions associated with the Hardware Setup File or Hardware Selector, the DSGetHWMgrErrorStatus routine (located in hwmgr.dll) should be used.

NOTE: The DSGetEngineErrorStatus call is actually inside Engine Manager (enmgr.dll). Note also that Engine Manager maintains an "error/status listbox" window which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of both status and error conditions, including both those resulting from DSPower series DLL calls and generated by DSP Engines. For complete information on error handling in Engine Manager, see section 7.5, Error Handling in Engine Manager.

Below is an example showing error investigation after a DSAssignCard failure:

```
if (!DSAssignCard(hEngine, szCardDesignator, uBusType, uIOBaseAddr, uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSAssignCard would be NULL in the case of failure, since a non-NULL value would be a valid card (hardware) handle. Thus, to find out more information, the DSGetEngineErrorStatus, DSGetEngMgrErrorStatus, and DSGetHWLibErrorStatus calls are checked. Note that DSGetEngineErrorStatus requires a valid engine handle as an input parameter. Thus, for problems related to a DSEngineOpen call, the DSGetEngineErrorStatus function should be avoided.

Below is a listing of error values returned by DSGetHWLibErrorStatus and DSGetEngineErrorStatus, along with a brief description of their meaning. These constants can be found in the hwlib.h include file; for a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in the Overview section of this chapter, Error and Status Conditions. Note that in C/C++ programs, it is safest to treat these error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, above, for more information about this grouping.

Hardware Library Error Values

DSDSPFILENOTFOUND

DSHARDWARENOTRESPONDING

DSINVALIDCARDHANDLE

DSCOULDNOTOPENTEMPLATEFILE

DSCOULDNOTCREATESOURCEFILE

DSCREATINGENGINEPROGRAMERROR

DSUNABLETOALLOCATEMEMORY

DSUNABLETOREADTEMPLATEFILE

DSERRORINDATAPARAM

DSCARDDOESNOTSUPPORTCALL

DSINVALIDWORDLENGTH

DSINVALIDMEMORYTYPE

DSINVALIDCARDDESIGNATOR

DSCARDNOTINHWSETUPFILE

DSALLCARDHANDLESALLOCATED

DSINVALIDCORENUMBER

Related Topics

General DLL Error and Status Conditions

Engine Error and Status Conditions

Hardware Manager Error and Status Conditions

Hardware Library Error and Status Conditions

Signal Processing Library Error and Status Conditions

General Topics

Error Handling in Engine Manager

Error Handling in Hardware Manager

Error/Status Window in Engine Manager

Error and Exception Handling in DirectCore Software

DSDSPFILENOTFOUND

Typically occurs after DSLoadFileCard or DSLoadFileCores call. Indicates a valid program (executable code) file could not be found.

DSHARDWARENOTRESPONDING

Typically occurs after a DSInitCard call. Indicates multicore CPU or DSP/analog hardware not present or not installed correctly.

DSINVALIDCARDHANDLE

Card handle was invalid.

DSCOULDNOTOPENTEMPLATEFILE

A template file needed by a high-level function could not be opened.

DSCOULDNOTCREATESOURCEFILE

A DSP Engine source code file being generated by a high-level function could not be created.

DSCREATINGENGINEPROGRAMERROR

An error occurred when compiling or linking a DSP Engine source code file(s).

DSUNABLETOALLOCATEMEMORY

Memory for the function could not be allocated

 Copyright © Signalogic 1994-2013

DSUNABLETOREADTEMPLATEFILE

An error occurred reading a template file needed b a high-level function.

DSERRORINDATAPARAM

A parameter passed to a high level function in a data structure was invalid.

DSCARDDOESNOTSUPPORTCALL

The hardware type does not support the call.

DSINVALIDWORDLENGTH

A wordlength parameter passed to a memory transfer function was invalid.

DSINVALIDMEMORYTYPE

A memory type parameter passed to a memory transfer function was invalid.

DSINVALIDCARDDESIGNATOR

The card designator string was not recognized, or not listed in the Hardware Setup File.

DSCARDNOTINHWSETUPFILE

The card handle or designator is no longer listed in the Hardware Setup File

DSALLCARDHANDLESALLOCATED

No more card handles can be allocated.

DSINVALIDCORENUMBER

The core number was out of range vs. the number of cores on the hardware.

Copyrght © Signalogic 1994-2013

6.2.5 Signal Processing Library Error and Status Conditions

This section not available yet.

Copyright © Signalogic 1994-2013

6.3 DSP Engine Error and Status Conditions

DSP Engine error and status conditions and text displays are generated and returned by programs such as Hypersignal-Macro series software or MATLAB, even though they may appear in status boxes or listboxes maintained by DSPower-Block Diagram software or a DSPower series DLL. For example, Engine Manager, which is part of DirectCore software, offers an error/status listbox which can be displayed as a debug tool, and which shows error and status conditions and text displays generated by DSP Engines as well as itself and other DSPower series DLLs. For more information, see Signalogic Software Users Guide section 7.5.3, Error/Status Window in Engine Manager.

Copyrght © Signalogic 1994-2013

6.3.1 Hypersignal Error and Status Conditions

This section not available yet.

Copyright © Signalogic 1994-2013

6.3.2 MATLAB Error and Status Conditions

This section not available yet.

 Copyrght © Signalogic 1994-2013

Glossary of Terms

API

Application Programming Interface.

ARM

Type of host CPU. Typically runs Linux.

COFF

Common Object File Format. Along with ELF format, one of two (2) file formats for executable code output by Texas Instruments build tools. For TI projects that include SYSBIOS v6 or higher, COFF format cannot be used.

CSL

Chip Support Library. A series of libraries and APIs offered by Texas Instruments to provide chip support and management functions such as cache, memory subsystem, PLL (clock rate), timers, etc. Typically each CSL library supports a chip family, for example C64x, C64x+, C66x.

DRAM

Dynamic random access memory. Typically used as large amount of auxiliary memory; for a multicore CPU or DSP/analog card, but sometimes used instead of SRAM if the DSP device has enough onchip SRAM. DRAM is typically 4-5 times slower, but much larger capacity per unit of circuit board area than SRAM.

ELF

Executable and Linkable Format. Along with COFF format, one of two (2) file formats for executable code output by Texas Instruments build tools. For TI projects including SYSBIOS v6 or higher, only ELF format is supported by the TI build tools.

EX

Extended.

extended memory

In legacy DOS and Windows systems, “extended memory” is system memory located above 1024 kbyte. This is the memory referred to when a system is described as having 4 Mbyte or RAM, or 16 Mbyte of RAM, etc. installed.

 Copyright © Signalogic 1994-2013

expanded memory

In legacy DOS and Windows systems, “expansion memory” is additional, usually large, amount of memory added to a PC system, but accessed through a small (64 kbyte) segment in conventional memory space. No longer as prevalent with advent of systems configured with 4 Mbyte or more of standard (extended) memory.

ISR

Interrupt service routine.

LKM

Loadable Kernel Module device driver (a type of Linux device driver).

NDA

Non-disclosure agreement.

NRT

Non-Real-Time.

PPC

PowerPC type of CPU.

RT

Real-Time

SRAM

Static random access memory. Typically used as main program and data memory for a multicore CPU or DSP/analog card. 4-5 times faster and more expensive than DRAM.

Talker

“Talker” refers to a special type of data transfer between host interface code and the multicore CPU or DSP hardware used due to limited port width, or “narrow pipe” hardware constraints; for example, if only certain types of memory on the hardware are visible to the host interface. This is sometimes the case with older, legacy hardware. In such cases the code resident on the hardware, whose job was to marshal and complete data transfers to non-host-accessible memory areas, came to be known as a “talker” program.

VxD

Virtual Device Driver. A type of Windows device driver used to perform low-level system functions or communication.

x86

Intel type of CPU.

.mak

Filename extension used for Microsoft Visual C/C++ v1.5 project files.

.tcf

Filename extension of SYSBIOS config files used by Texas Instruments Code Composer Studio software..

.prt

Filename extension of project files used by Texas Instruments Code Composer Studio v3.3 or lower.